AD__________________

Award Number: DAMD17-00-C-0031

TITLE: Modeling for Military Operational Medicine Scientific and Technical Objectives (Articulated Human Biomechanical Modeling Toolbox)

PRINCIPAL INVESTIGATOR: James H. Stuhmiller, Ph.D.
Weixin Shen

CONTRACTING ORGANIZATION: Jaycor, Incorporated
San Diego, California 92121-1002

REPORT DATE: December 2000

TYPE OF REPORT: Final, Phase I, Part II

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2000	Final, Phase I, Part II ( )

4. TITLE AND SUBTITLE
Modeling for Military Operational Medicine Scientific and Technical Objectives (Articulated Human Biomechanical Modeling Toolbox)

5. FUNDING NUMBERS
DAMD17-00-1-0031

6. AUTHOR(S)
James H. Stuhmiller, Ph.D.
Weixin Shen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Jaycor, Incorporated
San Diego, California 92121-1002

E-Mail:

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Report contains color.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for Public Release; Distribution Unlimited.	

13. ABSTRACT ***(Maximum 200 Words)***

14. SUBJECT TERMS	15. NUMBER OF PAGES
	139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

J00-3150.31-136

Articulated Human Biomechanical Modeling Toolbox

Phase I Report
Part II: Toolbox Routines

Prepared by:

Weixin Shen
Jaycor, Inc.
3394 Carmel Mountain Road
San Diego, California 92121-1002

Prepared for:

Commander
U.S. Army Medical Research and Materiel Command
Ft. Detrick, Maryland 21702-5012

Contract No. DAMD17-00-C-0031

December 2000

JAYCOR

TABLE OF CONTENTS

1. Introduction

This report documents the progresses on developing a human biomechanical modeling toolbox during the first phase of the research project. It is separated into two parts. A separate first part consists of an overview of the toolbox, rigid body formulations, and example models and applications. This second part provides detailed description of individual routines in the toolbox.

The second chapter describes the data format used in the toolbox. Data I/O, conversion and file format routines are listed in the chapter. A graphical editor for editing TMT files, TMTEDITOR, is also developed. Details on TMTEDITOR is given in Appendix A. Chapter three lists routines related to rigid body dynamics, including kinematic calculation, inverse dynamic analysis and forward dynamic simulation. Chapter four provides details of graphical routines, including routines related to the creation and manipulation of 3D graphical objects, user interface handling, animation and other graphical operation. Details on two graphical viewers, xyviewer and stickviewer, can be found in Appendix B and Appendix C respectively. Chapter five describes other utility routines related to mathematical calculation and string manipulation.

2

2. File Format and Data I/O Routines

Data Types of Toolbox Routines

Most data types of Matlab, including *double, char, cell,* and *struct* used to develop the toolbox routines. Detailed description of these data types can be found in matlab manual. The use of user defined data type (object) is intentionally avoided. This is because the current version of Matlab compiler does not support objects in generating standalone application, while one of the guidelines of developing the toolbox routines is the full support of generating standalone application.

Data types, *double, char, cell,* and *struct,* as supported in MATLAB, are simplified and customized for the development of toolbox routines. The simplification is made to facilitate the preparation of input and output data of the routines while maintaining the capability of handling complex data. Each data type supports single or multiple dimensional arrays. The details of the supported data types are given in Table 2-1.

Table 2-1. Data Types Used in Toolbox Routines

<table>
<tr><th colspan="3">Type</th><th>Example</th><th>Description</th></tr>
<tr><td rowspan="10">ARRAY</td><td rowspan="5">Double</td><td>Integer</td><td>10</td><td rowspan="5">Double precision numerical array. Notice that
1. Complex numbers are not supported
2. Arrary dimension is limited to two
3. A space in a numeric array indicates the following elements be put in row-wise
4. A semicolon (;) indicates the following elements be put column-wise</td></tr>
<tr><td>Real number</td><td>-10.1</td></tr>
<tr><td>Column vector</td><td>[1;2;3;4;5]</td></tr>
<tr><td>Row vector</td><td>[1 2 3 4 5]</td></tr>
<tr><td>Matrix</td><td>[1 2 3; 4 5 6]</td></tr>
<tr><td rowspan="3">String</td><td>A string</td><td>'this is a string'</td><td rowspan="3">Character arrays are put between two primes ('). Two consecutive primes indicates a prime inside a string

Curly brackets represent cell arrays. Only single dimensional cells of strings are supported</td></tr>
<tr><td>A string</td><td>'It''s good'</td></tr>
<tr><td>A string cell</td><td>{'string 1' 'string2' 'string 3'}</td></tr>
<tr><td rowspan="2">Structure</td><td>A simple structure</td><td>S.name = 'name'
S.data = [1 2; 3 4]</td><td rowspan="2">Structure arrays have field names. The fields contain other arrays, including structures. This is a very general data type that can collect related data and information together.

Only one dimensional structure array is supported</td></tr>
<tr><td>A more complex structure</td><td>S(2).name = 'name'
S(2).data.a = 1
S(2).data.b = [1 4]</td></tr>
</table>

Data Import and Export

An application program has to input and output data. The data may be imported from and exported to files, graphics user interfaces (GUIs), and other application programs. For example, a pre-processor usually inputs data from GUI and/or some descriptive files and generates data files for a solver. A solver may read from the files the solver parameters, time history data, tables, etc. and performs the calculation. The calculation results are usually saved as files or exported directory to post-processor for the analysis of results.

The toolbox is designed to have the capability of developing the whole application program from pre-processor to solver to post-processor, as well as the flexibility of being only part of the application program. Therefore, a common data interface supporting the data types used in the toolbox routines is essential. An application program developed from toolbox routines support the following three types of files

- MATLAB default binary file (MAT file) is the primary file format used to share data. This format supports all data types of matlab and is platform independent.
- Tagged matlab text file format (TMT file format) is developed as the ASCII counterpart of MAT file. TMT files support most data types used in toolbox routines with certain limitations. TMT files are used to share application parameters and time history data. The details on TMT files are given in section <Tagged Matlab Text Files>.
- Customized file formats. The toolbox also provides routines to interface with some customized file formats. The details on these file interface routines are described in section <Other File Format>

Tagged Matlab Text Files

Data Types Supported

The data types as described in Table 2-1 are supported.

Syntax

Structure array, as described in Table 2-1, is a data type with named *"data containers"* called *fields*. The fields of a structure can contain any type of data including *double, char, cell,* or another *structure array*. Therefore, structure allows storing dissimilar data according to their physical meaning and thus facilitates the data storage and reference among routines.

(a) TMT File (sample.tmt)

```
This is a sample TMT file

<NUM integer> 10;
<NUM number> -10.1;
<NUM col_vector> 1;
                 2;
                 3;
                 4;
                 5;
<NUM row_vector> 1 2 3 4 5;
<NUM matrix> 1 2;
             3 4;
<NUM thist> <TAB tdata col
1 4 5>
</NUM>
this is a comment line
<CHA char> ' a string'
<CHA cell> 'string 1'
           'string 2'
</CHA>
this is a comment line
<STR struct>
     <CHA name> 'name'
     <STR data>
          <NUM a> 1;
          <NUM b> 2;
     </STR>
</STR>
```

(b) Structure S

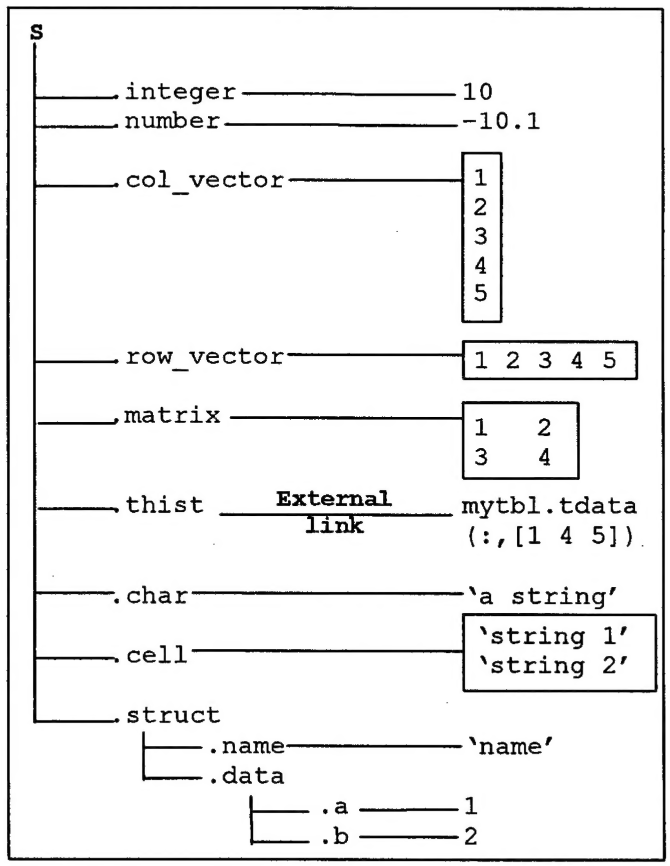


Figure 2-1 Example of a TMT file and the Corresponding Data

A TMT file, in essence, is the representation of a structure in an ASCII file. The *fields* of the structure are defined in the file by using a number of *tags* and simple syntaxes. Figure 2-1 gives an example of a TMT file and the structure it represents.

A TMT file ignores all line breaks, which means it is equivalent to write the whole file in one line or break it into hundreds of lines. As shown in Figure 2-1(a), a TMT files consists of three parts: *tags, contents* and *comments*. Tags include *Field Declaration Tags, Field Closure Tags,* and *External Link Tags*. The details of the use of tags are given in Table 2-2

Table 2-2. Syntax of Tag Components of a TMT file

Type	Syntax	Description
Field Declaration Tags	<STR *name(dim)*> or <STR *name*>	Declare that the following data are fields of the dim^{th} component of structure *name* until </STR> tag is met. Default dimension of one is assumed when *(dim)* is not present Must always be paired by </STR>
	<NUM *name*>	Declare the following *numeric contents* to be the value of *name* until a new <STR>, <NUM>, <CHA>, </STR>, </NUM>, and </CHA> is met
	<CHA *name*>	Declare the following *string cell contents* to be the value of *name* until a new <STR>, <NUM>, <CHA>, </STR>, </NUM>, and </CHA> is met
Field Closure Tags	</STR>	Close a structure declaration. Must always be used to pair <STR>.
	</NUM >	Used optionally to pair with <NUM>. Usually only required when comment after a <NUM> declaration
	</CHA>	Used optionally to pair with < CHA >. Usually only required when comment after a < CHA > declaration
External Link Tags	</TAB *name* COL *id* > or </TAB *name* ROW *id* >	Used as part of the *contents* after <NUM> declaration to load numerical matrices (tables) from an external data structure. *name* is the field name of the external data structure where the data is to be loaded COL or ROW indicates load data column-wise or row-wise *id* is the indice of the rows or columns to be loaded

Notice the following rules apply to the tags.

- Only the first three characters of tag keywords **STRucture, NUMerics, CHAracter, TABle, COLumn,** and **ROW,** are discriminated. All the following characters are ignored.
- Tag keywords are case insensitive
- Field name declaration inside any tag is case sensitive

Two types of contents, i.e., *numeric content* and *string cell content* are described in Table 2-3.

Table 2-3. Syntax of Tag Components of a TMT file

Type	Example		Description
Numeric Content	Number	10	A space inside a numeric content indicates the following data be collocated column-wisely A semicolon (;) indicates the following data be collocated column-wisely The size of matrix must match when input numerical content
	Column vector	1;2;3;4;5;	
	Row vector	1 2 3 4 5;	
	Matrix	1 2; 3 4;	
String Cell Content	String	'string'	Any string content must be put inside a pair of primes (') A prime inside a string must be indicated by two consecutive primes ('') Strings and string cells are always saved in data type *cell of string* when loaded
	String	'It''s a string'	
	String cell	'string 1' 'string 2' 'string 3'	

Any contents outside a *Field Declaration Tag* and its corresponding *Field Closure Tag* is treated as comments and is ignored. Notice in order to add comments after a <NUM> or a <CHA> tag, optional </NUM> or </CHA> tag must be used.

An Example of Using Tmt Files

Two interpreting routines are developed. ***tmt2struct*** reads a TMT file and converts it into a structure. ***struct2tmt*** saves a structure into a TMT file.

An example is given to shown the use of these routines and load data from external structures. First, save the TMT file as *'sample.tmt'*. In order to load this file, an external structure, say *mytbl* with a field named *tdata* must exist. The external structure can be generated from another TMT file, say *'data.tmt'* as follows

DATA File (data.tmt)

```
This is a sample data file

<NUM tdata>
    11    12    13    14    15    16    17    18    19;
    21    22    23    24    25    26    27    28    29;
    31    32    33    34    35    36    37    38    39;
    41    42    43    44    45    46    47    48    49;
    51    52    53    54    55    56    57    58    59;
    61    62    63    64    65    66    67    68    69;
    71    72    73    74    75    76    77    78    79;
    81    82    83    84    85    86    87    88    89;
    91    92    93    94    95    96    97    98    99;
</NUM>

<NUM otherdata> 1
<CHA otherchar> 'char'  'string'
```

Run **mytbl = tmt2struct('data.tmt')** to generate a structure *mytbl* with field *mytbl.tdata* being a 9 by 9 matrix as listed above. Then run **S = tmt2struct('test.tmt',mytbl)** to generate the structure *S* as given in Figure 2-1.

The *S.thist* is loaded from the [1 4 5] columns of the external *mytbl.tdata* field, i.e.,

```
S.thist = [
    11    14    15
    21    24    25
    31    34    35
    41    44    45
    51    54    55
    61    64    65
    71    74    75
    81    84    85
    91    94    95
];
```

Finally run **struct2tmt(S,'test_out.tmt')** to generate another TMT file *'test_out.tmt'* that is equivalent to the original *'test.tmt'* file but with external data *mytbl.tdata* built-in. The *'test_out.tmt'* is given as follows.

```
<NUM integer> 10;
<NUM number> -10.1;
<NUM col_vector> 1;
                 2;
                 3;
                 4;
                 5;
<NUM row_vector> 1 2 3 4 5;
<NUM matrix> 1 2;
             3 4;
<NUM thist> 11 14 15;
            21 24 25;
            31 34 35;
            41 44 45;
            51 54 55;
            61 64 65;
            71 74 75;
            81 84 85;
            91 94 95;
<CHA char> ' a string'
<CHA cell> 'string 1'
           'string 2'
<STR struct>
     <CHA name> 'name'
     <STR data>
          <NUM a> 1;
          <NUM b> 2;
     </STR>
</STR>
```

TMTEDITOR

TmtEditor is developed to brower and edit TMT files. Details on TMTEDITOR is given in Appendix A.

Other File Format

ASCII Data Files

The AHBM toolbox supports common ASCII data file formats, such as space or tab delimited files (*.txt, .dat); comma delimited files (.csv); etc. Routines are developed to convert data among file formats.

GDIF File Format

General Data Interchange Format (GDIF) is a self-documented ASCII format (.jif) with variable name, units, and description included within the file to record time-traces. The GDIF ASCII format can be converted into a binary form (.jib). The GDIF binary format can be accessed by the specialized programs developed by Jaycor, Inc.

StdMat Data File

StdMat is a customized Matlab binary file (*.mat) format to record matrix data. Each variable (array) in the file is a structure with data saved column-wisely in a matrix. The variables should have the fields as given in Table 2-4.

Table 2-4. Fields of a Variable in StdMat File

Field Name	Description
name	A string vector with each element being the name of one column of matrix data
label	A string vector with each element being the label (additional comments) of one column of matrix data
units	A string vector with each element being the units of one column of matrix data
val	Matrix data (saved column-wisely)
groupname	The name of the group under which the data is grouped

StdMat file formats provides a common ground where complex matrix data (including time traces) can be saved and shared. It can be accessed by the I/O functions in Matlab and the I/O routines developed in AHBM toolbox.

INI File

INI file is the window initialization file format (*.ini). INI files are mostly used for developing GUI applications.

Data Conversion and I/O Routines

List of Data Conversion and I/O Routines

tmt2struct:	load a structure from tmtfile (and possibly a table file)
struct2tmt:	save structure to a tmtfile (and possibly a table file)
ini2struct:	read a window ini file and save the data as a struct
struct2ini:	save struct data into window ini file
mat2stdmat:	matrix to std structured format conversion
stdmat2csv:	save std structure format as a csv file
stdmat2jif:	save std structure format as a jif file
jif2stdmat2:	read a jif file in std structured format
load_ascii:	the counterpart of 'load filename -ascii' in standalone

tmt2struct

SYNOPSIS

S = tmt2struct(tmtfile,TABLE)

INPUTS

Tmtfile:	Tagged Matlab Text filename
TABLE:	(optional) external structure referred from tmtfile

OUTPUT

S:	the structure loaded from tmtfile

DESCRIPTION

TMT2STRUCT load a structure from tmtfile. If tmtfile also refers to external data

EXAMPLES

First generate a tmtfile and a tablefile using struct2tmt

```
S.name = 'sample string'
S.data = rand(100,3);
struct2tmt(S,'test.tmt','test.table');
```

Read external data from tablefile 'test.table'

```
mydata = tmt2struct('test.table');
```

Read S from 'test.tmt' and mydata

```
S = tmt2struct('test.tmt',mydata);
```

NOTE

ROUTINES CALLED

A number of internal functions

SEE ALSO

struct2tmt

struct2tmt

SYNTAX

struct2tmt(S,tmtfile,tablefile,option,desp);

INPUT:

S: the structure to be output
tmtfile : Tagged Matlab Text filename
tablefile: (optional) the filename of an additional table file where very long data of txtfile is stored and cross-referred
option: (optional) 'replace' (default) or 'add'

OUTPUT

none

DESCRIPTION

STRUCT2TMT save a structure in a TMT file

EXAMPLES

S.name = 'sample string'
S.data = rand(100,3);
struct2tmt(S, 'test.tmt'); will save structure S to test.tmt
struct2tmt(S,'test.tmt','test.table'); will save structure to 'test.tmt'

The S.data will be saved in 'test.table' with a <TAB ...> link created in 'test.tmt'

NOTE

1. When tablefile is not input, all data will be saved in txtfile. When tablefile is input, all numeric data with size greater than 50 will be saved in the tablefile, and a cross-reference <TAB ...> will be added in the txtfile
2. Set option = 'replace' will overwrite txtfile or tablefile if they are already exist. Set option = 'add' will append to the existing files

ROUTINES CALLED

A number of internal functions

SEE ALSO

tmt2struct

ini2struct

SYNOPSIS

S = ini2struct(file);

INPUTS

File: file name, the file should follow the above format the window ini file should follow the following convention

[section name]
varname1 = value1
varname2 = value2

currently, section name line is ignored value should be number, row vector, or a string

OUTPUT

S: the structure loaded from an INI file

DESCRIPTION

INI2STRUCT reads an INI file and saves the data as a structure

EXAMPLES

First generate an INI file

```
S.dir1 = 'c:\';
S.data = [1 1 2 3];
struct2ini('try.ini',S);
```

Then read from the INI file

```
T = ini2struct('try.ini');
```

NOTE

Only row vectors can be used as numerical value

ROUTINES CALLED

none

SEE ALSO

struct2ini

struct2ini

SYNOPSIS

struct2ini(file,S);

INPUTS

File: file name, the file should follow the above format. The window INI file should follow the following convention

[section name]
varname1 = value1
varname2 = value2

Currently, section name line is ignored. Value should be number, row vector, or a string

S: structure to be output to the INI file

OUTPUT

none

DESCRIPTION

STRUCT2INI writes an INI file from a structure

EXAMPLES

```
S.dir1 = 'c:\'
S.data = [1 1 2 3];
struct2ini('try.ini',S);
```

NOTE

Only row vectors can be used as numerical input

ROUTINES CALLED

none

SEE ALSO

ini2struct

mat2stdmat

SYNOPSIS

stdmat = mat2stdmat(mat,name,label,units,groupname);

INPUTS

mat: matrix data/ string data
names:{ncol} cell or a single string/cell corresponding to each column data. If a single string is used, name_icol will be set for each column data
label: {ncol} cell or a single string/cell to each column data. If a single string is used, label_icol will be set for each column data
units: {ncol} cell or a single string/cell corresponding to each column data. If a single string is used, same units will be added to each column data
groupname : (optional) a single string, indicates the groupname of the matlab data

OUTPUT

stdmat: the standard structured mat data loaded from the file

DESCRIPTION

MAT2STDMAT converts a column-wise matrix into standard structured matlab data (StdMat) file.

EXAMPLES

```
V = rand(30,3);
stdV = mat2stdmat(V,{'V_1','V_2','V_3'},{'V_1','V_2','V_3'},...
{'m','m','m'})
```

NOTE

ROUTINES CALLED

A number of internal functions

SEE ALSO

stdmat2csv, stdmat2jif

stdmat2csv

SYNOPSIS

stdmat2csv(csvfile,stdmat,option);;

INPUT:

csvfile:	name of the csv file,
stdmat:	data follows the standard structured mat format
option:	'add' or 'replace' for adding to the file or rewrite the file. The default value for option is 'add'

OUTPUT:

A CSV file where the data will be saved row-wisely

DESCRIPTION

STDMAT2CSV writes a CSV ASCII file from the standard structured matlab data.

EXAMPLES

```
V = rand(30,3);
stdV = mat2stdmat(V,{'V_1','V_2','V_3'},{'V_1','V_2','V_3'},...
{'m','m','m'});
stdmat2csv('try.csv',stdV);
```

NOTE

ROUTINES CALLED

SEE ALSO

stdmat2jif

stdmat2jif

SYNOPSIS

stdmat2jif(gdif,stdmat);

INPUT:

gdif: name of the jif file, *.jif file extension will be added automatically
stdmat: structured mat data to be output

OUTPUT:

A GDIF ASCII file

DESCRIPTION

STDMAT2JIF writes a GDIF ASCII file from the standard structured matlab data.

EXAMPLES

```
V = rand(30,3);
stdV = mat2stdmat(V,{'V_1','V_2','V_3'},{'V_1','V_2','V_3'},...
{'m','m','m'});
stdmat2jif('try.jif',stdV);
```

NOTE

ROUTINES CALLED

A number of internal functions

SEE ALSO

stdmat2csv

jif2stdmat

SYNOPSIS

function DATA = jif2stdmat(gdif);

INPUTS:

Gdif: name of the GDIF file,

OUTPUT:

DATA: a structure contains StdMat structures as fields
.stdmat1
.stdmat2 ... etc

DESCRIPTION

JIF2STDMAT reads a GDIF file and saves it in the structure DATA. Each field in DATA is a StdMat structure

EXAMPLES

```
V = rand(30,3);
stdV = mat2stdmat(V,{'V_1','V_2','V_3'},{'V_1','V_2','V_3'},...
{'m','m','m'});
stdmat2jif('try.jif',stdV);
stdVin = jif2stdmat('try.jif')
```

NOTE

ROUTINES CALLED

A number of internal functions

SEE ALSO

stdmat2jif

load_ascii

SYNTAX

[data,errormsg] = load_ascii(filename);

INPUT:

filename:	the ASCII data file name

OUTPUT

data:	data matrix loaded from the ascii file
errormsg:	errormsg saves the error message if error is encountered in reading the file

DESCRIPTION

LOAD_ASCII is the counterpart of 'load filename -ascii' in standalone applications. It reads the first line of the ASCII file to get the number of columns of and then fast reads the ASCII file

EXAMPLES

ROUTINES CALLED

SEE ALSO

3. Rigid Body Routines

Kinematics Routines

List of Kinematics Routines

eul2r:	Euler angles to rotation matrix conversion
r2eul:	Rotational matrix to Euler angles conversion
ep2r:	Euler parameters to rotation matrix conversion
r2ep :	Rotational matrix to Euler parameters conversion
r2_body_ang:	Body orientation (Euler) angles calculation from its rotation matrix
r2_jnt_ang :	Joint angle calculation from rotation matrices

eul2r

SYNOPSIS

varargout = eul2r(eul,cnvt,opt)

INPUTS

eul: euler angles (3x1)
cnvt: Convention for euler angles ('zxz', 'zyx')
opt: (optional) vector (7x1) deciding which terms to be calculated
opt(1): calcaulte R, saved as 3x3 matrix or a nx9 matrix
opt(2): dR/deul(m), saved as 9x3 matrix in Rij,m
opt(3): dR^2/deul(m)/deul(n), saved as 9x6 matrix in Rij,mn
opt(4): S1 (3x3 matrix), omegab = S1*deul/dt
opt(5): T, (3x3 matrix), deul/de = T*omegab
opt(6): dT/deul(m), saved as 9x3 matrix in Tij,m
opt(7): S2 (3x3 matrix), as in the equation
S1*deul^2/dt^2+S2*[de1*de2;de1*de3;de2*de3];

OUTPUT

varargout: matrices output dependent on opt

DESCRIPTION

EUL2R performs basic calculations regarding Euler angles to rotation matrix conversion such as calculating the rotational matrix R, its derivatives dR/de(m), dR^2/de(m)/de(n), the derivative to angular velocity matrix S, its inverse matrix T, its deriviative dT/de(m), and deriviative to angylar acceleration matrix S2

EXAMPLES

example # 1: calculate only R matrix
```
eul = [3 1 2];
R = eul2r(eul,'zxz');
```

example # 2: calculate; differential matrices; velocity matrix, etc
```
[R,dRdm,dRdmn,T,dTdm] = eul2r(eul,'zyx',[1 1 1 1 1]');
[T,dTdm] = eul2r(eul,'zyx',[0 0 0 1 1]);
```

example # 3: acceleration conversion matrix
```
S2 = eul2r(eul,'zyx',[0 0 0 0 0 0 1]);
```

example # 4: time series of rotation matrix
```
eul = [ 3 1 2; 2 1 1; 1 0 3; 0 0 pi/2];
R = eul2r(eul,'zxz');
```

NOTE

1. Using *reshape(dRdm(:,i),3,3)* to restore dRdm as a 3x3 matrix

2. when only R is calculated, vectorized programming is supported; R can be saved as a 3x3 matrix or a nx9 representing different frames
3. trailing zeros in opt can be neglected

ROUTINES CALLED

none

SEE ALSO

r2eul

r2eul

SYNOPSIS

eul = r2eul(R,cnvt,opt)

INPUTS

R: Rotational matrix R = [i,j,k]; (3x3 or nx9)
cnvt: Convention for euler angles ('zxz', 'zyx')
opt: options determining the default range of theta (default =1)
=1, theta = [0,pi] for ZXZ and [-pi/2 pi/2] for ZYX
=2, theta = [-pi,0] for ZXZ and [-pi,-pi/2] and [pi/2,pi] for ZYX

OUTPUT

eul: calculated Euler angles

DESCRIPTION

R2EUL calculates Euler angles from a rotation matrix (3x3) or a series of rotation matrices (nx9)

EXAMPLES

```
eul = [3 4 2];
R = eul2r(eul,'zxz');
eul1 = r2eul(R,'zxz');
eul2 = r2eul(R,'zxz');
```

NOTE

A small number is used to judge if gimble locking occurs, the number del=1e-5

ROUTINES CALLED

none

SEE ALSO

eul2r

ep2r

SYNOPSIS

varargout = ep2r(ep,opt)

INPUTS

ep: uler parameters (4x1)
opt: etermine whether to calculate each term (5x1)
opt(1)=1: calculate R , saved as 3x3 matrix
opt(2)=1: calculate dR/de(m) , saved as 9x4 matrix in Rij,m
opt(3)=1: calculate dR^2/de(m)/de(n), saved as 9x10 matrix in Rij,mn
opt(4)=1: calculate T (4x3) as in de= T*w_b
opt(5)=1: calculate dT/de(m) (saved as Tij,m 12x4)

OUTPUT

varargout: matrices output dependent on opt

DESCRIPTION

EP2R: performs basic calculations regarding Euler parameters to rotation matrix conversion such as calculating the rotational matrix R, its derivatives dR/de(m), dR^2/de(m)/de(n), the derivative to angular velocity matrix T, and its derivative dT/de(m)

EXAMPLES

example # 1: calculate only R matrix

```
ep = [3 1 2 10];
R = ep2r(ep);
```

example # 2: calculate R matrix; differential matrices, etc

```
[R,dRdm,dRdmn,T,dTdm] = ep2r(ep,[1 1 1 1 1]');
[T,dTdm] = ep2r(ep,[0 0 0 1 1]);
```

NOTE

1. Default of option calculates only R, opt = [1 0 0 0 0];
2. Trailing zeros in opt can be neglected

ROUTINES CALLED

none

SEE ALSO

r2ep

r2ep

SYNOPSIS

ep = r2ep(R)

INPUTS

R: rotational matrix (3x3)

OUTPUT

ep: euler parameters (4x1)

DESCRIPTION

R2EP calculates the Euler parameters from a rotational matrix

EXAMPLES

```
eul = [3 4 2];
R1 = eul2r(eul,'zxz');
ep = r2ep(R1);
R2 = ep2r(ep);
```

NOTE

ROUTINES CALLED

none

SEE ALSO

ep2r

r2_body_ang

SYNOPSIS

ang = r2_body_ang(R,type,ang0);

INPUTS

R: rotational matrix (3x3)
type: convention ('zxz','zyx')
ang0: Initial euler angles (usually the value of previous time step)

OUTPUT

ang: body orientation (Euler) angles (3x1),

DESCRIPTION

R2_BODY_ANG calculates the Euler angles of a given body relative to the default coordinate system. If ang0 is also given, the ang will start from ang0. This enables the range of Euler angles be extended beyond [-pi pi] for tumbling motion;

EXAMPLES

```
eul = [3 4 2];
R = eul2r(eul,'zxz');
ang1 = r2_body_ang(R,'zxz');
ang2 = r2_body_ang(R,'zxz',[2.5 3.5 1.9]);
```

notice in the example, ang2 is exactly same as eul; while ang1 is not

NOTE

If ang0 is given, the program will automatically
- Eliminate the jump due to degeneracy
- Add or remove 2*n*pi to make solution continuous

ROUTINES CALLED

r2eul

SEE ALSO

r2_jnt_ang

r2_jnt_ang

SYNOPSIS

jang = r2_jnt_ang(R1,R2,type,jang0);

INPUTS

R1: rotational matrix for 1st segment
R2: rotational matrix for 2nd segment
type: type of joints
jang0: initial joint angles (usually the value of previous time step)

OUTPUT

jang: calculated joint angles

DESCRIPTION

R2_JNT_ANG calculates the joint angles given two rotational matrices for the two segments connecting the joint. If jang0 is also given, the jang will start from jang0. This extends the range of joint angles beyond [-pi pi] and allows the tracking of tumbling motion

EXAMPLES

```
eul1 = [3 4 2];
eul2 = [3 4 2];
R1 = eul2r(eul1,'zxz');
R2 = eul2r(eul2,'zxz');
jang1 = r2_jnt_ang(R1,R2,'zxz');
jang2 = r2_jnt_ang(R1,R2,'zxz',[0 2*pi 0]);
```

NOTE

Currently, zxz, zyx, pin, null2d, pin3d joints are supported

ROUTINES CALLED

r2eul

SEE ALSO

r2_body_ang

Forward Dynamics Routines

List of Forward Dynamics Routines

fwd_simu	main setup routine for forward dynamics analysis
fwd_integrator	ode integrator setup routine
fwd_equation	forward dynamics ode equation routine
projection	constraint projection routine
cnstode45	ode 45 non-stiff solve for constrained system
cnstode15s	ode 15 stiff solve for constrained system
cnstode23s	ode 23 stiff solve for constrained system
jnt_cnst	G, g1g2 and g due to joint constraints
jnt_cnst_euler	G, g1g2 and g due to Euler joint constraints
jnt_cnst_pin	G, g1g2 and g due to 3d pin joint constraints
jnt_cnst_pin2d	G, g1g2 and g due to 2d pin joint constraints
jnt_cnst_null2d--	G, g1g2 and g due to 2d null joint constraints
jnt_react	joint reaction force calculation
jnt_react_euler	joint reaction force due to an Euler joint
jnt_react_pin	joint reaction force due to a 3d pin joint
jnt_react_pin2d	joint reaction force due to a 2d pin joint
jnt_react_null2d	joint reaction force due to a 2d null joint
spring_force	calculate spring force
damper_foce	calculate damping force
stop_force	calculate joint soft stop force
ddRxppl	calculate vR2 = Rij,mn*dpm*dpn*lj
dRxl	calculate R1l = Rij,m*lj
dTxpv	calculate the vector vR2 = Tlm,n*dpn*vm

fwd_simu

SYNOPSIS

t_cpu = fwd_simu(job_file, choice);

INPUTS

job_file:	file keep job information (include the system to use)
choice:	select the task to perform

input':	read in all job, model and force files
'initialization':	check default, error etc, setup geometry
'run':	run simulation, sorting data ...
'all':	perform all the preceding tasks (default)

OUTPUT

t_cpu:	cpu time for the task

GLOBAL:

SYSTEM	system description structure
BODY	body description structures
JOINT	joint description structures
JOB	job description structure
EXF	external force structures

DESCRIPTION

FWD_SIMU set up a forward dynamics model by

- read job and ahm input files
- verify the input data are correct
- setup the initial configuration
- setup the geometry patch
- run integration

EXAMPLES

NOTE

ROUTINES CALLED

A number of internal routines
fwd_integrator

SEE ALSO

fwd_integrator

SYNOPSIS

[time,Y,STAT] = fwd_integrator(options);

INPUTS

options:	extra options for the integrator (refers to odeset)

OUTPUT

time:	time vector when solution is outputed
Y:	position and velocity solution
STAT:	solver statistics

GLOBAL:

SYSTEM	system description structure
BODY	body description structures
JOINT	joint description structures
JOB	job description structure
EXF	external force structures

DESCRIPTION

FWD_INTEGRATOR sets up the ode integrator for forward dynamics problem, performs the integration, and saves the results in result and restart files

EXAMPLES

NOTE

ROUTINES CALLED

fwd_equation

SEE ALSO

fwd_equation

SYNOPSIS

varargout = fwd_equation(t,y,flag,varargin);

INPUTS

t:	time	
y:	[p,v]' (variable of the 1st order ODE system)	
flag:	flag of task to be performed	
	'':	(default) evaluate y' = f(y)
	'update':	update solution after a successful step
	'call_proj':	call projection routine for position and velocity constraints
	'proj_1':	called from projection routine to calculate M, G and gi
	'proj_2':	called from projection routine to calculate g
varargin:	other input arguments to be passed on, including	
	isproj:	=1 do project; =0 do not project
	isupdate:	=1 update solution after a successful step, =0 do not

OUTPUT

varargout: variable outputs depend on the flag

DESCRIPTION

FWD_EQUATION setup the forward dynamics equations for ODE solver

EXAMPLES

NOTE

ROUTINES CALLED

projection, cnstode45, cnstode15s, cnstode23s

SEE ALSO

projection

SYNOPSIS

ypproj = projection(odefile,t,y,NP,NV,NL);

INPUT:

odefile:	filename of the ode and constraint formulation
t:	current time
y:	original converged solution
NP:	number of position degrees of freedom
NV:	number of velocity degrees of freedom
NL:	number of constraints equations

OUTPUT:

yproj:	projected solutions

OUTPUT

time:	time vector when solution is outputed
Y:	position and velocity solution
STAT:	solver statistics

DESCRIPTION

PROJECTION projects the approximation solution back to the position and velocity constraint manifolds

EXAMPLES

NOTE

ROUTINES CALLED

cnstode45, cnstode15s, cnstode23s

SEE ALSO

cnstode45

SYNOPSIS

[tout,yout,varargout] = ode15s(odefile,tspan,y0,options,varargin)

INPUTS

refer to ode45

OUTPUT

refer to ode45

DESCRIPTION

CNSTODE45 is the extension of ODE45 non-stiff ode solver to include position and velocity constraints

EXAMPLES

NOTE

ROUTINES CALLED

none

SEE ALSO

cnstode15s; cnstode23s

cnstode15s

SYNOPSIS

[tout,yout,varargout] = ode15s(odefile,tspan,y0,options,varargin)

INPUTS

refer to ode15s

OUTPUT

refer to ode15s

DESCRIPTION

CNSTODE15S is the extension of ODE15S stiff ode solver to include position and velocity constraints

EXAMPLES

NOTE

ROUTINES CALLED

none

SEE ALSO

cnstode45; cnstode23s

cnstode23s

SYNOPSIS

[tout,yout,varargout] = ode23s(odefile,tspan,y0,options,varargin)

INPUTS

refer to ode23s

OUTPUT

refer to ode23s

DESCRIPTION

CNSTODE23S is the extension of ODE23S stiff ode solver to include position and velocity constraints

EXAMPLES

NOTE

ROUTINES CALLED

none

SEE ALSO

cnstode45; cnstode15s

jnt_cnst

SYNOPSIS

varargout = jnt_cnst(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
jnt: structure data for the joint
opt: option of calculation
1. calculate the contribution to g
2. calculate the contribution to G
3. calculate the contribution to G and gagb

OUTPUT

varargout: output depends on opt

DESCRIPTION

JNT_CNST calculates the contribution of the joint constraints to G, g1g2 and g

EXAMPLES

NOTE

ROUTINES CALLED

Joint constraint routines for various joints

SEE ALSO

jnt_cnst_euler

SYNOPSIS

varargout = jnt_cnst_euler(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
jnt: structure data for the joint
opt: option of calculation
1. calculate the contribution to g
2. calculate the contribution to G
3. calculate the contribution to G and gagb

OUTPUT

varargout: output depends on opt

DESCRIPTION

JNT_EULER_CNST calculates the contribution of an Euler joint to G, g1g2 and g

EXAMPLES

NOTE

An Euler joint only involves position constraint. When the joint is connected to the ground, the position should equal to the designated position. Otherwise, the two neighboring bodies are connected at the joint

R*l - Og = 0;
R1*l1 - R2*l2 = 0;

ROUTINES CALLED

ddRxppl, dTxpv, drxl

SEE ALSO

jnt_cnst_pin

SYNOPSIS

varargout = jnt_cnst_pin(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
jnt: structure data for the joint
opt: option of calculation
1. calculate the contribution to g
2. calculate the contribution to G
3. calculate the contribution to G and gagb

OUTPUT

varargout: output depends on opt

DESCRIPTION

JNT_CNST_PIN calculates the contribution of a 3D pin joint to G, g1g2 and g

EXAMPLES

NOTE

A pin joint involves position constraint (as in an Euler joint) pluses two rotational constraints

ROUTINES CALLED

ddRxppl, dTxpv, drxl; joint_cnst_euler

SEE ALSO

jnt_cnst_pin2d

SYNOPSIS

varargout = jnt_cnst_pin(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
jnt: structure data for the joint
opt: option of calculation
1. calculate the contribution to g
2. calculate the contribution to G
3. calculate the contribution to G and gagb

OUTPUT

varargout: output depends on opt

DESCRIPTION

JNT_CNST_PIN2D calculates the contribution of a 2D pin joint to G, g1g2 and g

EXAMPLES

NOTE

A pin2d joint only involves position constraint. When the joint is connected to the ground, the position should equal to the designated position, otherwise, the two neighboring bodies are connected at the joint

R*l - Og = 0;
R1*l1 - R2*l2 = 0;

ROUTINES CALLED

ddRxppl, dTxpv, drxl

SEE ALSO

jnt_cnst_null2d

SYNOPSIS

varargout = jnt_cnst_null2d(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
jnt: structure data for the joint
opt: option of calculation
1. calculate the contribution to g
2. calculate the contribution to G
3. calculate the contribution to G and gagb

OUTPUT

varargout: output depends on opt

DESCRIPTION

JNT_CNST_PIN2D calculates the contribution of a 2D null joint to G, g1g2 and g

EXAMPLES

NOTE

No constraint is involved for a 2D null joint

ROUTINES CALLED

SEE ALSO

jnt_reaction

SYNOPSIS

f = jnt_reaction(t,P,V,body1,body2,joint);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
joint: structure data for the joint

OUTPUT

f: calculated reaction force

DESCRIPTION

JNT_REACTION calculates the joint reaction forces due to joint spring, damper or joint soft stop

EXAMPLES

NOTE

ROUTINES CALLED

Joint reaction force routines for various joints

SEE ALSO

jnt_react_euler

SYNOPSIS

f = jnt_react_euler(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
joint: structure data for the joint

OUTPUT

f: calculated reaction force

DESCRIPTION

JNT_REACT_CNST calculates the Euler joint reaction forces due to joint spring, damper or joint soft stop

EXAMPLES

NOTE

An Euler joint has three rotational degree of freedom ground can only be inboard

ROUTINES CALLED

r2_jnt_ang; spring_force; damper_force; stop_force

SEE ALSO

jnt_react_pin

SYNOPSIS

f = jnt_react_pin(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
joint: structure data for the joint

OUTPUT

f: calculated reaction force

DESCRIPTION

JNT_REACT_PIN calculates the 3D pin joint reaction forces due to joint spring, damper or joint soft stop

EXAMPLES

NOTE

ROUTINES CALLED

r2_jnt_ang; spring_force; damper_force; stop_force

SEE ALSO

jnt_react_pin2d

SYNOPSIS

f = jnt_react_pin2d(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
joint: structure data for the joint

OUTPUT

f: calculated reaction force

DESCRIPTION

JNT_REACT_PIN2D calculates the 2D pin joint reaction forces due to joint spring, damper or joint soft stop

EXAMPLES

NOTE

ROUTINES CALLED

r2_jnt_ang; spring_force; damper_force; stop_force

SEE ALSO

jnt_react_null2d

SYNOPSIS

varargout = jnt_react_null2d(t,P,V,body1,body2,jnt,opt);

INPUTS

t: current time
P: position vector
V: velocity vector
body1: structure data for inboard body
body2: structure data for outboard body
joint: structure data for the joint

OUTPUT

f: calculated reaction force

DESCRIPTION

JNT_REACT_NULL2D calculates the 2D null joint reaction forces due to joint spring, damper or joint soft stop

EXAMPLES

NOTE

ROUTINES CALLED

r2_jnt_ang; spring_force; damper_force; stop_force

SEE ALSO

spring_force

SYNOPSIS

F = spring_force(type,prop,d);

INPUTS

type: type of spring (nspring x 1) cell
- 'linear': linear spring represented by k
- 'tabular': nonlinear spring represented by tabular form

prop : spring properties data cell
- 'linear': k
- 'tabular': [d(:) F(:)]

d: joint relative displacement (nspring x 1), d should be in ascending order

OUTPUT

F: spring force (nspring x 1)

DESCRIPTION

SPRING_FORCE calculates the spring forces according to the type and properties of the spring

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

damper_force

ddRxppl

SYNOPSIS

vR2 = ddRxppl(dRdmn,dp,l)

INPUTS

dRdmn:	dR^2/dp(m)/dp(n)
l:	postion vector
dp:	dp/dt

OUTPUT

vR2:	resultant vector

DESCRIPTION

ddRxppl calculates the vector vR2 = Rij,mn*dpm*dpn*lj

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

dRxl

SYNOPSIS

R1l = drxl(dRdm,l)

INPUTS

dRdm:	dR/dp(m)
l:	postion vector

OUTPUT

Rl1:	calculated matrix R1l; 3x3 for Euler angles; 3x4 for Euler parameters

DESCRIPTION

dRxl calculates the matrix R1l = Rij,m*lj

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

dTxpv

SYNOPSIS

vT = dTxpv(dTdm,dp,v)

INPUTS

dTdm:	as in pdot = dTdm *v, see zxz2t
dp:	dp/dt
v:	segment angular velocity

OUTPUT

vT:	calculated vector

DESCRIPTION

dTxpv calculates the vector vR2 = Tlm,n*dpn*vm

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

Inverse Dynamics Routines

List of Inverse Dynamics Routines

inv_analysis	main setup routine for inverse dynamics analysis
inv_kinematics	kinematics calculation
inv_dynamics	inverse dynamics calculation

inv_analysis

SYNOPSIS

[SYSTEM,JOB,BODY,JOINT,EXF] = inv_analysis(jobfile);

INPUTS

jobfile: file keep job information (include the model system to use)

OUTPUT

SYSTEM	system description structure
BODY	body description structures
JOINT	joint description structures
JOB	job description structure
EXF	external force structures

DESCRIPTION

INV_ANALYSIS performs the following tasks:

1. verify and read in the job, model and data
2. kinematics analysis
   - 2.1 filter the kinematics data (body, joint)
   - 2.2 calculate linear velocity and acceleration
   - 2.3 calculate body and joint angles
   - 2.4 calculate angular velocity and acceleration
   - 2.5 calculate joint angle
3. dynamics analysis
   - 3.1 Calculate the joint forces and torques in gloval frame
   - 3.2 Convert the force and torque into body local frame
   - 3.3 Convert the force and torque into anatomical frame
4. Calculate additional energetic quantities
5. Output results to files

EXAMPLES

NOTE

ROUTINES CALLED

A number of internal routines
inv_kinematics,
inv_dynamics
math function; i/o functions, etc

SEE ALSO

inv_kinematics

SYNOPSIS

[BODY,JOINT] = inv_kinematics(SYSTEM,JOB,BODY,JOINT);

INPUTS

SYSTEM:	SYSTEM definition structure
JOB:	JOB definition structure
BODY:	BODY definition structure
JOINT:	JOINT definition Structure

OUTPUT

BODY:	BODY definition structure, with updated kinematics information
JOINT:	JOINT definition structure, with updated kinematics information

DESCRIPTION

INV_KINEMATICS performs the following tasks:

1. filter the kinematics data (body, joint)
2. calculate linear velocity and acceleration
3. calculate body and joint angles
4. calculate angular velocity and acceleration
5. calculate joint angle

EXAMPLES

NOTE

ROUTINES CALLED

matfiltfilt:	Butterworth filtering of matrix data
dxdt:	derivative sof uniformly spaced data
r2_body_ang:	body orientation (Euler) angle calculation
eul2r:	euler angle to rotation matrix conversion

SEE ALSO

inv_dynamics

inv_dynamics

SYNOPSIS

JOINT = inv_dynamics(SYSTEM,JOB,BODY,JOINT,EXF);

INPUTS

SYSTEM:	SYSTEM definition structure
JOB:	JOB definition structure
BODY:	BODY definition structure
JOINT:	JOINT definition Structure
EXF:	External force data structure

OUTPUT

JOINT:	JOINT definition structure, with updated dynamics information

DESCRIPTION

INV_DYNAMICS performs the following tasks:

1. calculate the joint forces and torques in the global frame
2. convert the force and torque into an anatomical frame

EXAMPLES

NOTE:

This routine works for an open-loop (tree) model, where a body can have more than one proximal joints, but only one distal joints

ROUTINES CALLED

SEE ALSO

inv_kinematics

4. Graphical Routines

Graphical Objects Routines

List for graphical objects Routines

Create graphical objects

gen_patch_block:	generate a 3d block patch
gen_patch_cylinder:	generate a 3d cylindrical patch
gen_patch_sphere:	generate a 3d spherical patch
gen_patch_arrow:	generate an arrow patch
gen_patch_spring:	generate a spring patch
gen_patch_ground:	generate a patch representing the ground
read_patch_asc:	read a patch from an ASCII ASC file
read_patch_xix:	read a patch from an ASCII XIX file

Manipulate of graphical objects

affine_patch:	perform affine transformation of a patch
scale_patch:	scale a patch
add_patch_prop:	add additional graphical properties to a patch

gen_patch_block

SYNOPSIS

p = gen_patch_block(l,m,n,varargin)

INPUTS

l:	number of x elements
m:	number of y elements
n:	number of z elements
varargin:	parameter/value pairs to specify additional properties of the patch

OUTPUT

p:	geometrical patch object

DESCRIPTION

GEN_PATCH_BLOCK generates a 3D block object with unit length in all the x, y, and z directions. The center of the block is located at the origin.

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

gen_patch_cylinder

SYNOPSIS

p = gen_patch_cylinder(m,n,varargin)

INPUTS

m:	an even number of elements along circumference (default = 20)
n:	an even number of elements in longitudinal direction (default = 20)
varargin:	parameter/value pairs to specify additional properties of the spring

OUTPUT

p:	geometrical cylinder object

DESCRIPTION

GEN_PATCH_CYLINDER generates a cylindrical patch object of unit diameter and unit length and located at the origin and aligned in the z direction

EXAMPLES

NOTE

ROUTINES CALLED

add_patch_prop

SEE ALSO

gen_patch_sphere

SYNOPSIS

p = gen_patch_sphere(n,varargin)

INPUTS

n:	an even number of elements (default = 20)
varargin:	parameter/value pairs to specify additional properties of the sphere

OUTPUT

p:	geometrical spherical object

DESCRIPTION

GEN_PATCH_SPHERE generates a spherical patch object of unit diameter with its center located at the origin of the reference system

EXAMPLES

NOTE

ROUTINES CALLED

add_patch_prop

SEE ALSO

gen_patch_arrow

SYNOPSIS

p = gen_patch_arrow(P1,P2,lHead,wHead,wTail,varargin)

INPUTS

P1:	coordinates of the end of the arrow
P2:	coordinates of the tip of the arrow
lHead:	ratio of head length
wHead:	ratio of head width
wTail:	ratio of tail width
varargin:	parameter/value pairs to specify additional properties of the arrow

OUTPUT

p:	geometrical arrow object

DESCRIPTION

GEN_PATCH_ARROW generates a 3D geometrical object representing an arrow.

EXAMPLES

NOTE

ROUTINES CALLED

add_patch_prop

SEE ALSO

gen_patch_spring

SYNOPSIS

p = gen_patch_spring(P1,P2,m,w,width,varargin)

INPUTS

P1:	coordinates of starting point
P2:	coordinates of ending point
m:	number of rings in the spring
w:	width of the spring
varargin:	parameter/value pairs to specify additional properties of the spring

OUTPUT

p:	geometrical spring object

DESCRIPTION

GEN_PATCH_SPRING generate a 3D geometrical object representing a spring.

EXAMPLES

NOTE

ROUTINES CALLED

add_patch_prop

SEE ALSO

gen_patch_ground

SYNOPSIS

p = gen_patch_ground(m,n,color1,color2,varargin)

INPUTS

m:	number of checked squares in x direction
n:	number of checked squares in y direction
color1:	color one of checked squares, default=[0.2 0.2 0.2]
color2:	color two of checked squares, default=[0 0 0]
varargin:	parameter/value pairs to specify additional properties of the ground

OUTPUT

p:	geometrical ground object

DESCRIPTION

GEN_PATCH_GROUND generates a graphical patch object representing the ground. The ground is represented by checked interlacing squares. The patch is in XY plane with unit length in X and Y direction. The center is at the origin of the reference frame

EXAMPLES

NOTE

ROUTINES CALLED

add_patch_prop; alias2rgb

SEE ALSO

read_patch_asc

SYNOPSIS

asc = read_patch_asc(ascfile);

INPUTS

ascfile: asc path data file name

OUTPUT

Asc: geometrical patch structure with the following fields

Vertices:	coordinates of geometrical nodes
Faces:	node connectivity matrix
VertexNormals:	(optional) normal at the nodes (for graphical rendering)

DESCRIPTION

READ_PATCH_ASC reads in a geometrical patch defined in an ASCII ASC file

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

read_patch_xix

read_patch_xix

SYNOPSIS

xix = read_patch_xix(xixfile);

INPUTS

xixfile: xix data file name

OUTPUT

xix: geometrical patch structure with the following fields
- Vertices: coordinates of geometrical nodes
- Faces: node connectivity matrix
- VertexNormals: (optional) normal at the nodes (for graphical rendering)

DESCRIPTION

READ_PATCH_XIX reads in a geometrical patch defined in an ASCII XIX file

EXAMPLES

NOTE

```
FORMAT of an xix file:
line 1:         comment
line 2:         NDIM
line 3:         is_std_ix, node_per_face
line 4:         nVertices, nFaces
one comment line
                vertices coordinates
one comment line
                IX data
Normal (optional)
                node normal data
CData (optional)
                FaceVerticeCData
FaceColor (optional)
                face color
EdgeColor (optional)
                edge color
```

ROUTINES CALLED

SEE ALSO

read_patch_asc

affine_patch

SYNOPSIS

pa = affine_patch(p,tran,R)

INPUTS

p:	geometrical patch object (usually aligned along the default coordinate system)
tran:	translation along the x, y, z axes
R:	rotational matrix representing orientation of the patch in the coordinate system

OUTPUT

pa:	geometrical patch object after affine transformation

DESCRIPTION

AFFINE_PATCH performs an affine transformation of a geometrical patch

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

scale_patch

SYNOPSIS

ps = scale_patch(p,scale)

INPUTS

p: geometrical patch object (usually aligned along the default coordinate system)

scale: scaling factor of the patch in x,y,z axes

OUTPUT

ps: scaled patch

DESCRIPTION

SCALE_PATCH scales a geometrical patch

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

add_patch_prop

SYNOPSIS

newP = add_patch_prop(oldP,varargin)

INPUTS

oldP:	old geometrical patch object
varargin:	parameter/value pairs to specify additional properties of the arrow

OUTPUT

newP:	new geometrical patch object

DESCRIPTION

ADD_PATCH_PROP adds/modifies parameter/value pairs of a geometrical patch object

EXAMPLES

NOTE

Currently the following parameters are supported

EdgeColor
FaceColor
LineStyle
LineWidth

ROUTINES CALLED

SEE ALSO

User Interface Routines

List of User Interface Routines

geticoncdata:	read from an icon file the cdata (color map)
seticoncdata:	set cdata on a toolbar button
show_btn_ctxMenu:	associate an context menu to a toolbar button
enableiconcdata:	enable or disable a toolbar button
msgOutput:	message output routine
filterUI:	update filter type in a filter popup menu
axis2fig:	copy and re-scale a axis onto a figure
setpopupvalue:	set the value of a popup to match a given string

geticoncdata

SYNOPSIS

cdata = geticoncdata(iconfile,idx,bgcolor);;

INPUTS

iconfile:	name of a icon file
idx:	(default=1) the number of icon in the icon file
bgcolor:	the bg color to set as transparent

OUTPUT

cdata:	RGB color data matrix of the icon

DESCRIPTION

GETICONCDATA reads from an icon file and save the icon as cdata. If bgcolor is provided, it also attempts to save the bgcolor as NaN. When used with seticoncdata, bgcolor will be displayed transparent

EXAMPLES

NOTE

The program can be modified to include alpha data (transparency)

ROUTINES CALLED

SEE ALSO

seticoncdata

SYNOPSIS

seticoncdata(h,Cdata);

INPUTS

h:	handle of the obj (pushbutton, etc)
Cdata:	n x m x 3 color data

OUTPUT

cdata:	RGB color data matrix of the icon

DESCRIPTION

SETICONCDATA sets the CData on a UI (pushbutton etc). All NaN components will be displayed as the UI background color (looks like transparent)

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

show_btn_ctxMenu

SYNOPSIS

show_btn_ctxMenu;

INPUTS

none

OUTPUT

none

DESCRIPTION

SHOW_BTN_CTXMENU displays context menu associated with a tool button. The handle of the tool button should be saved as the *userdata* of the button and the *enable* of the tool buttonn should be set as *'inactive'* the *buttondownfcn* of the button should be set as *'show_btn_ctxMenu'*

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

enableiconcdata

SYNOPSIS

enableiconcdata(hbtn,option);

INPUTS

hbtn:	handle of the obj (pushbutton, etc) (may be a vector)
option:	'enable' or 'disable'

OUTPUT

cdata:	RGB color data matrix of the icon

DESCRIPTION

enableiconcdata: enable or disable a tool button

EXAMPLES

NOTE

The use of multiple handles (hbtn being a vector) is supported

ROUTINES CALLED

SEE ALSO

msgOutput

SYNOPSIS

msgOutput(msg)

INPUTS

msg: a string or a cell or strings (the message)

OUTPUT

none

DESCRIPTION

MSGOUTPUT outputs the message in the *msg* string to a command window, a message GUI window and/or a message file

EXAMPLES

example one -- output message to command window
```
msgOutput('message to command window');
```

example two -- output message to msgwindow and save in a message file (tmp.msg)
```
close all; set(gcf,'unit','pixels')
h = uicontrol('style','listbox','tag','MsgWindow','pos',[10 10
200 100],'max',100);
setappdata(h,'msgFile','tmp.msg');
msgOutput({'example of msg output','also check the
tmp.msg file'});
```

NOTE

1. msgOutput first look for a msgwindow with a the tag of 'MsgWindow' (case senstive) if the msgwindow is not present, the msg will be output to the command window; otherwise the message will be added to the message window.
2. the maximum number of lines of message can be specified by setting the 'max' property of the UI control of the message window
3. the msg will be save as the appdata 'MSG' in msg
4. if appdata 'msgFile' is present in the message window, the msg will aslo be saved in the file

ROUTINES CALLED

SEE ALSO

filterUI

SYNOPSIS

filterUI(h,type);

INPUTS

h: the filter UI handle (a popup menu);
type: string of the type of filter

OUTPUT

none

DESCRIPTION

filterUI updates the types of filter displayed in a popup menu and automatically set the value according to the input type string

EXAMPLES

```
close all; set(gcf,'unit','pixels');
h = uicontrol('style','popupmenu','pos',[100 100 200
20],'string','filter example');
filterUI(h,'2nd order Butterworth');
filtertype = popupstr(h)
```

NOTE

To get the filter type from the UI, use *popupstr*

ROUTINES CALLED

SEE ALSO

axis2fig

SYNOPSIS

hnew = axis2fig(hold)

INPUTS

hold: original handle of the axis to be copies

OUTPUT

hnew: the handle of the new figure

DESCRIPTION

axis2fig copies all visible components on a axis to a new figure, so all components can be re-scaled to normal size to be printed

EXAMPLES

```
h = axes('unit','pixel','pos',[0 0 100 100]);plot(1:10); legend('plot x');
h = axis2fig(h);
```

NOTE

The position of legend will be auto put in one of the four corners

ROUTINES CALLED

SEE ALSO

setpopupvalue

SYNOPSIS

setpopupvalue(h,s);

INPUTS

h: handle of popup or listbox
s: string to be matched

OUTPUT

none

DESCRIPTION

SETPOPUPVALUE sets the value of popup or listbox to match the specified string. Exact match of lower case is required

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

Animation, Viewers, and Other

List of Animation, Viewers and Other Routines

alias2rgb:	convert an alias of a color to RGB color
anim_dyn_1st:	generate the 1st animation frame for an inverse or a forward dynamic model
anim_dyn_ith:	generate the ith animation frame for an inverse or a forward dynamic model
read_asf:	read an ASCII TekScan data file
anim_asf:	animate pressure data measured by TekScan
xyviewer:	see Appendix B
stickviewer:	see Appendix C

alias2rgb

SYNOPSIS

rgb = char2rgb(c)

INPUTS

c: character symbol of a color

OUTPUT

rgb: rgb representation of the color

DESCRIPTION

ALIAS2RGB converts a color alias to RGB color

EXAMPLES

NOTE

Alias of colors supported are listed as follows

y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

ROUTINES CALLED

SEE ALSO

anim_dyn_1st

SYNOPSIS

h = anim_dyn_1st(SYSTEM,BODY);

INPUTS

SYSTEM:	system description structure
BODY:	body description structure

OUTPUT

h:	handles of graphical objects representing the bodies

DESCRIPTION

ANIM_DYN_1st draws the first frame of an inverse or a forward dynamical model given the model description and time trace of model response. It also sets up the axis property.

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

anim_dyn_ith

SYNOPSIS

h = dyn_anim_ith(iframe,time,SYSTEM,BODY);

INPUTS

iframe:	the number of the frame to be displayed
time:	time vector
SYSTEM:	system description structure
BODY:	body description structure

OUTPUT

h:	handles of graphical objects representing the bodies

DESCRIPTION

ANIM_DYN_ith draws the i^{th} frame of an inverse or a forward dynamic model

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

read_asf

SYNOPSIS

[INFO,P] = read_asf(asffile,maxframe);

INPUTS

asffile:	ascii tekscan data file
maxframe:	(option) max. number of frames to read from the file default is to read all the frames

OUTPUT

INFO: information structure of the tekscan data with the following fields
'sensor_type'
'rows'
'cols'
'units'
'row_spacing'
'row_spacing_units'
'col_spacing'
'col_spacing_units'
'noise_threshold'
'scale_factor'
'exponent'
'seconds_per_frame'
'movie_filename'
'start_frame'
'end_frame'

P: pressure data saved as a cell, each cell element is a matrix of data (rows x cols)

DESCRIPTION

read_asffile reads an ASCII Tekscan data file

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

anim_asf

anim_asf

SYNOPSIS

M = anim_asf(INFO,P);

INPUTS

INFO: information structure of the tekscan data with the following fields
'sensor_type'
'rows'
'cols'
'units'
'row_spacing'
'row_spacing_units'
'col_spacing'
'col_spacing_units'
'noise_threshold'
'scale_factor'
'exponent'
'seconds_per_frame'
'movie_filename'
'start_frame'
'end_frame'

P: pressure data saved as a cell, each cell element is a matrix of data (rows x cols)

OUTPUT

M: matlab movie data from the animation

DESCRIPTION

anim_asf generates the animation of a set of Tekscan test data

EXAMPLES

NOTE

ROUTINES CALLED

SEE ALSO

read_asf

5

5. Utilities Routines

Math Routines

List of Math Routines

cross2d:	2D cross product
isint:	check if input numerical variable is integer
isrealnum:	check if input numerical variable is real
unit:	normalize a matrix
dxdt:	calculate time derivatives of a uniformly spaced signal
matfiltfilt:	filter uniformly spaced signal with a double Butterworth filter
power_spec:	calculate power spectrum of a time-domain signal
r_times_v:	rotate 2D or 3D vectors

cross2d

SYNOPSIS

c = cross2d(a,b);

INPUT:

a:	a 2D vector
b:	a 2D vector

OUTPUT:

c:	the cross procuct (a number)

DESCRIPTION

CROSS2D calculates the cross product of two 2D vectors

EXAMPLES

```
c1 = cross2d([1 2],[1 2]);
c2 = cross2d([10 0],[1 1]);
```

NOTE

Cross product is not commutative, which means the result depends on the sequence of the two vectors

ROUTINES CALLED

SEE ALSO

isint

SYNOPSIS

status = isint(a,asize)

INPUT:

a: number to be checked
asize: (optional) size of a to be expected

OUTPUT:

status: 1 true; 0 for false

DESCRIPTION

ININT check if *a* is a numerics integer. The size of *a* can also be checked

EXAMPLES

isint(a) check if all elements of 'a' is integer
isint(a,[1 1]) check if 'a' is a integer scalar
isint(a,[0 1]) check if 'a' is a integer column vector
isint(a,[0 2]) check if 'a' is a integer max with 2 columns
isint(a,[1 0]) check if 'a' is a integer row vector
isint(a,[2 0]) check if 'a' is a integer max with 2 rows
isint(a,[4 6]) check if 'a' is a integer max of size 4x6

NOTE

Use zero to indicate the length of a row or a column can be variable

ROUTINES CALLED

SEE ALSO

isrealnum

isrealnum

SYNOPSIS

status = isrealnum(a,asize)

INPUT:

a: number to be checked
asize: (optional) size of a to be expected

OUTPUT:

status: 1 true; 0 for false

DESCRIPTION

isrealnum checks if *a* is a numerical real value. The size of *a* can also be checked

EXAMPLES

isrealnum(a) check if all elements of 'a' is real
isrealnum(a,[1 1]) check if 'a' is a real number
isrealnum(a,[0 1]) check if 'a' is a real column vector
isrealnum(a,[0 2]) check if 'a' is a real max with 2 columns
isrealnum(a,[1 0]) check if 'a' is a real row vector
isrealnum(a,[2 0]) check if 'a' is a real max with 2 rows
isrealnum(a,[4 6]) check if 'a' is a real max of size 4x6

NOTE

use zero to indicate the length of a row or a column can be variable

use the isrealnum to avoid conflict with builtin isreal function

ROUTINES CALLED

SEE ALSO

isint

unit

SYNOPSIS

U = unit(A,dim)

INPUT:

A: matrix data
dim: option of perfroming the calculation
dim=0 make the matrix a unit matrix
dim=1 make every column of the matrix a unit vector
dim=2 make every row of the matrix a unit vector

OUTPUT:

U: output matrix data

DESCRIPTION

UNIT normalizes the input matrix or its column or row vectors

EXAMPLES

```
A = rand(10,4)
Umatrix = unit(A,0);
Ucol = unit(A,1)
Urow = unit(A,2)
```

NOTE

ROUTINES CALLED

SEE ALSO

dxdt

SYNOPSIS

xn = dxdt(X,dt,order);

INPUTS

X: sample, X can be a vector, matrix or a 3D matrix
dt: sampling spacing
order: the order of derivative (1 or 2)
xn: deriviate

OUTPUT

xn: nth order derivative of original data

DESCRIPTION

DXDT calculates the n^{th} derivatives of X. X should be uniformly sampled with a spacing dt. If X is a 2D or 3D matrix, it is differentiated column-wisely. *order* is one or two with default being one

EXAMPLES

```
X = rand(100,5);
dx = dxdt(X,0.1,1);
ddx = dxdt(X,0.1,2);
```

NOTE

1. forward difference is used for the 1st element; backward difference is used for the last element; and central difference is used for all the other
2. 1st and last element of the second order derivatives are the linear exterpolation of the neighboring values

ROUTINES CALLED

SEE ALSO

matfiltfilt

SYNOPSIS

xf = matfiltfilt(dt, fcut, N, X);

INPUT:

dt: sampling rate
fcut: cutoff frequency (Hz) fcut must <= nyquist freq
N: order of the filter (usually 2 or 4)
X: sample, X can be a column vector, a matrix or a 3d matrix

OUTPUT:

xf: filtered data

DESCRIPTION

MATFILTFILT filters a uniform input signal in time domain by a lower-pass double Butterworth filter of specified order

EXAMPLES

```
X = rand(100,1);
xf = matfiltfilt(0.01,10,2,X);
plot(1:100,X,'r:',1:100,xf);
legend('original signal','filtered signal');
```

NOTE

fcut must be smaller than nyquist freqency (1/dt/2)

ROUTINES CALLED

butter: in matlab/signal toolbox

SEE ALSO

power_spec

SYNOPSIS

```
[fs,Freq,Power] = power_spec(T,X);
```

INPUT:

T: uniformly spaced time vector
X: input signal in time domain

OUTPUT:

fs: samping frequency
Freq: frequence vector
Power: Output power spectrum

DESCRIPTION

POWER_SPEC calculates power spectrum of input signal in time domain by performing fast Fourier transformation

EXAMPLES

```
T = 1:100;
X = rand(1,100);
[fs,Freq,Power] = power_spec(T,X);
plot(Freq,Power);
```

NOTE

Frequency is shift by half the Nyquist frequency to make it symmetric

ROUTINES CALLED

SEE ALSO

r_times_v

SYNOPSIS

V = r_times_v(R,v);

INPUT:

R: 2x2, nx4 (2D time trace), 3x3, nx9 (3D time trace) matrix
v: a length of 2 or 3 vector, or nx2 (2D time trace), nx3 (time trace)

OUTPUT:

V: rotated vector(s)

DESCRIPTION

R_TIMES_V rotates a 2D or 3D vector or its time traces by the times the vector with a 2D or 3D rotational matrix or its time traces

EXAMPLES

Example #1

```
R = [1 1 1; 2 2 2; 3 3 3];
v = [1 2 3]';
V = r_times_v(R,v);
```

Example #2 (for time trace, R is put columnwise)

```
R = [1 1 1 2 2 2 3 3 3
     1 0 3 4 5 0 1 3 2];
v = [1 2 3
     0 0 1];
V = r_times_v(R,v);
```

NOTE

ROUTINES CALLED

SEE ALSO

String Routines

List of String Manipulation Routines

parchar:	use partial of ASCII table to remove preceding and trailing blanks, tabs, special characters, etc
sepchar:	separate a string into a cell array (blank, tab delimited)
dirDirs:	get the name of all directories in a designated directory
dirFiles:	get the name of all files in a designated directory
isdir:	determine if the specified directory exists
isfile:	determine if the specified file exists
addFileExt:	check and add a designated extension to a file
isvar_wdof:	determines if a string is a valid variable name. Array DOF may be included
str2realmat:	convert a string matrix to a numeric real matrix
fieldparts:	separate full field name (from top structure to field) into structure path and field name
fullfield:	construct the full field name (from top structure to field) from the structure path and fieldname

parchar

SYNOPSIS

s = parchar(s)

INPUT:

s: input string

OUTPUT:

s: output string with only characters with ascii table 33-125

DESCRIPTION

PARCHAR uses only the partial ASCII character (32-125) table of matlab, s can be a string or a string cell. Preceding and trailing spaces and tabs are also eliminated; tabs inside the text is converted into spaces

EXAMPLES

```
parchar('   a b');
parchar({'cha 1','  cc 2'});
```

NOTE

ROUTINES CALLED

SEE ALSO

sepchar

sepchar

SYNOPSIS

c = sepchar(s)

INPUT:

s: input string

OUTPUT:

c: output cell of strings

DESCRIPTION

SEPCHAR separates a character *'s'* into a cell, with each element corresponds to the part of character separated by space, tab, etc

EXAMPLES

```
sepchar('   1  3 4');
```

NOTE

ROUTINES CALLED

SEE ALSO

parchar

dirDirs

SYNOPSIS

d = dirDirs(p,option);

INPUT:

p: directory (default is the current directory)
option: option = 1: ignore '.' and '..'
option = 0, '.' and '..' will be included

OUTPUT:

d: directory names saved in a cell

DESCRIPTION

dirDirs get the subdirectories in a directory

EXAMPLES

```
dirDirs
dirDirs('c:\')
```

NOTE

ROUTINES CALLED

SEE ALSO

dirFiles

dirFiles

SYNOPSIS

f = dirFiles(p);

INPUT:

p: directory (default is the current directory)

OUTPUT:

f: all filenames in the directory saved in a cell

DESCRIPTION

dirFiles gets the files in a directory

EXAMPLES

```
dirFiles
dirFiles('c:\')
```

NOTE

ROUTINES CALLED

SEE ALSO

dirDirs

isdir

SYNOPSIS

result = isfile(dirname)

INPUT:

dirname:	the name of a directory

OUTPUT

result:	=1 directory exists; =0 directory does not exist

DESCRIPTION

ISDIR checks if *dirname* is a directory

EXAMPLES

ROUTINES CALLED

SEE ALSO

isfile

isfile

SYNOPSIS

result = isfile(filename)

INPUT:

filename:	the name of a file

OUTPUT

result:	=1 file exists; =0 file does not exist

DESCRIPTION

ISFILE checks if *filename* is a file

EXAMPLES

ROUTINES CALLED

SEE ALSO

isdir

addFileExt

SYNOPSIS

fname = addFileExt(filename,ext);

INPUT:

filename:	input filename
ext:	file extension to be added(not dot)

OUTPUT:

fname:	filename with extension added

DESCRIPTION

ADDFILEEXT checks if the designated extension is in the *filename* and, if not, adds the designated extension to the file name

EXAMPLES

fname = addFileExt('fname.','.txt')

NOTE

1. the dot in extension does not matter
2. lower or upper case is neglected

ROUTINES CALLED

fileparts

SEE ALSO

isvar_wdof

SYNOPSIS

[status,var,dof] = isvar_wdof(c)

INPUT:

c:	variable name to be checked

OUTPUT:

status:	1 if is a valid name, 0 not
var:	the variable name
dof:	dof of the variable

DESCRIPTION

ISVAR_WDOF determines if c is a valid variable name. A valid variable name must start with a letter or _ and contains no special or blank characters. DOF of the variable can be included with (idof) after the variable name.

EXAMPLES

NOTE

ROUTINES CALLED

isvarname

SEE ALSO

str2realmat

SYNOPSIS

[mat,status] = str2realmat(s)

INPUT:

s: input string matrix or a cell with each element a string row

OUTPUT:

mat: output numerical array

status: 1: successful;

0: error in string matrix; (size doesn't match, NaN present)

DESCRIPTION

STR2REALMAT converts a string matrix to a numeric array, which is the extension of str2double and str2num

EXAMPLES

NOTE

ROUTINES CALLED

sepchar;

SEE ALSO

fieldparts

SYNOPSIS

[fpath,f] = fieldparts(ff)

INPUT:

ff: full structure field name

OUTPUT:

fpath: structure field path
f: field name

DESCRIPTION

FIELDPARTS separates full field name (from top structure to field) into the structure path and field name

EXAMPLES

[sp,f] = fullfield('S.S1.S(2).field1')

NOTE

ROUTINES CALLED

SEE ALSO

fullfield

fullfield

SYNOPSIS

ff = fullfield(fpath,f)

INPUT:

fpath: structure field path
f: field name

OUTPUT:

ff: full structure field name

DESCRIPTION

FULLFIELD constructs the full field name (from top structure to field) from the structure path and fieldname

EXAMPLES

ff = fullfield('S.S1.S(2)','field1')

NOTE

ROUTINES CALLED

SEE ALSO

fieldparts

A

A. TmtEditor

Introduction

What is TmtEditor

TmtEditor is a GUI based tool to browser and edit TMT files. Refer to Data I/O section for details on TMT files.

Features

- Easy browsing of complicated structured data
- Support generating new TMT files using existing template files
- Various ways of inputting data
- Flexible control over user's accessibility to data editing

Start TmtEditor

TmtEditor is delivered in one of the following three versions

- **MEX version**: MEX version of **TmtEditor** is to be used in Matlab environment. To start it in Matlab, type *'tmteditor'* in Matlab command window.

Note: To use TmtEditor in Matlab, the right path and default setting has to be setup for the directory where TmtEditor MEX routines are installed. To setup the default setting (for first time use or when the default setting is corrupt, type *'tmtsetup'* in Matlab command window

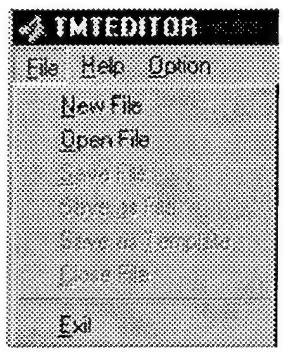
- **Standalone version:** Standalone version of **TmtEditor** is delivered in a single installation file **"install_TmtEditor.exe"**, which can be installed and run as a standard DOS/Window executable program.
- **Application version: TmtEditor** can also be integrated as part of application software as a data viewer. In this case, it can only be used with the software.

GUI Components

This section describes several GUI components of **TmtEditor** and their common use.

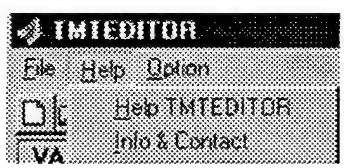
File Menu

File menu performs file operations



Help Menu

Help menu provides access to help information

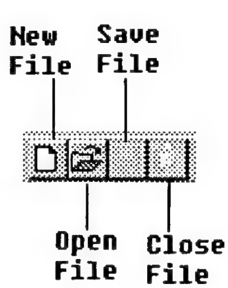


Option Menu

Option menu allows the user to select different accessibility to data editing

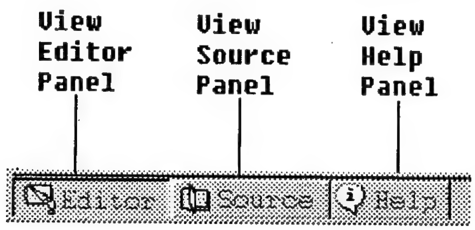
TmtEditor Toolbar

TmtEditor Toolbar provides easy access to common file operations:

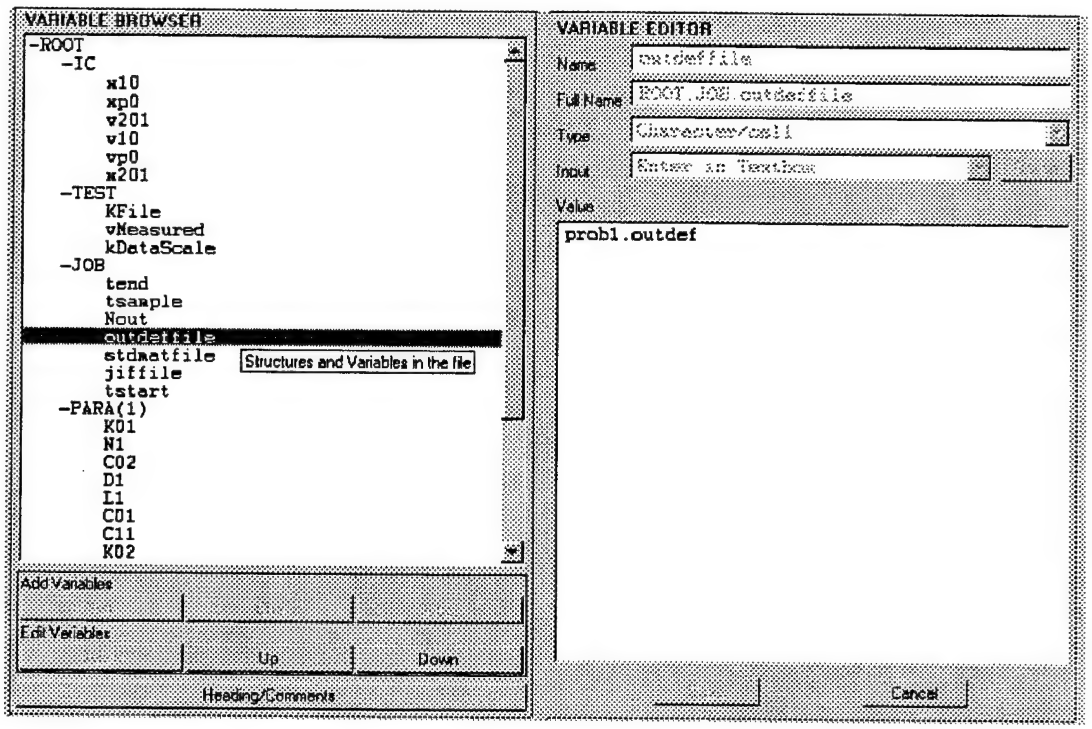


TmtEditor Viewbar

TmtEditor Viewbar allows the change of different view panels



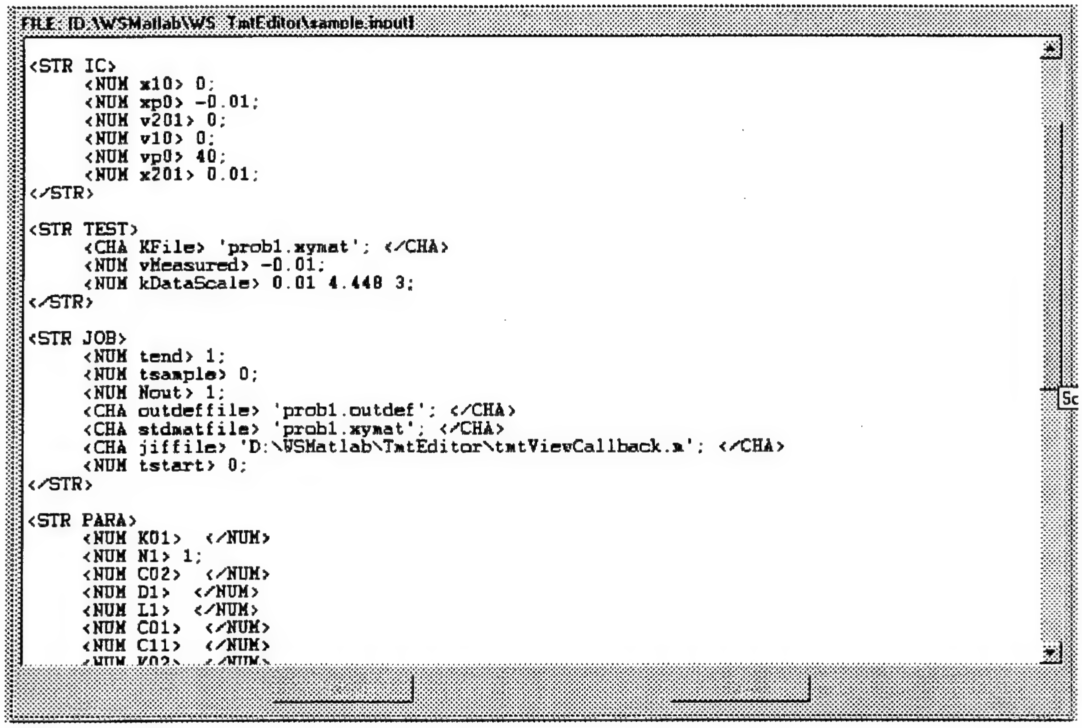
Editor Panel



Editor Panel includes a **Data Browser,** a **Variable Editor,** and other UI components and allows the browsing and editing of TMT files

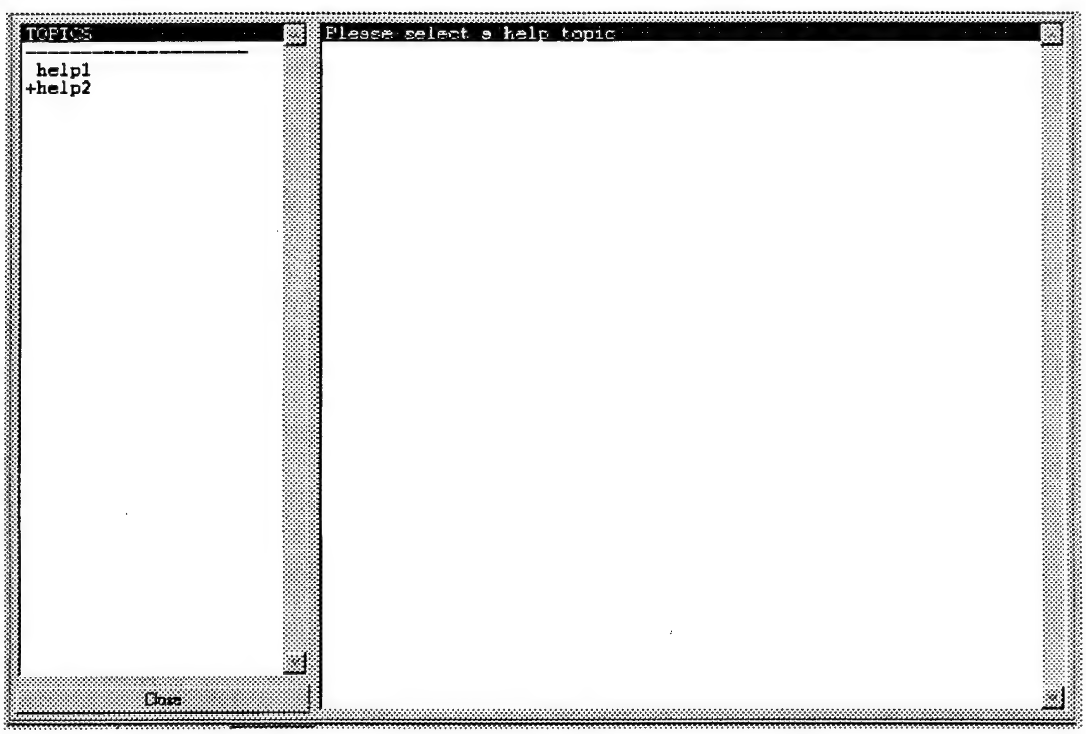
Source Panel

Source panel lists the source code of the file being edited.



Help Panel

Help Panel provides the browsing of help information on **TmtEditor**



Use TmtEditor

Change View Panel

To change the view panel, click on the buttons on the **TmtEditor Viewbar.**

File Operation

To open a TMT file, follow the following steps

- Select **Open File** from **File Menu;**

  or push the **Open File** button on **TmtEditor Toolbar;**
- Browse for the file to open in the popup **File Browse Window**

To close a data file, follow the following steps

- Select **Close File** from **File Menu;**

  or push the **Close File** button on **TmtEditor Toolbar;**
- If the file has be modified, a popup window will show up asking for saving or discarding the changes.

To save modifications to a TMT file,

- Select **Save File** from **File Menu;**

  or push the **Save File** button on **TmtEditor Toolbar;**

To save the file as another TMT file,

- Select **Save as File** from **File Menu;**
- Browse or enter the name for the new file in the popup **File Browse Window**

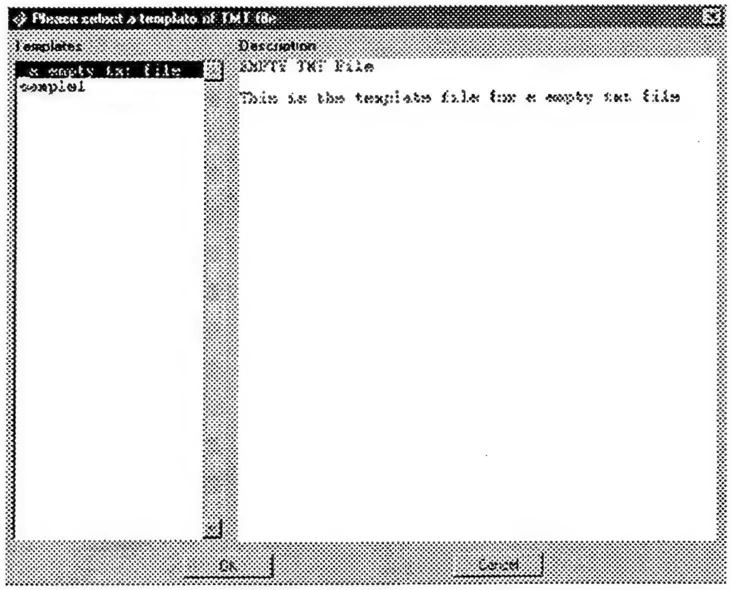
To save the file as an TMT template file,

- Select **Save as Template** from **File Menu;**
- Enter the name for the template file in the popup **File Browse Window**

To create a new TMT file

- Select **New File** from **File Menu;**

  or push the **New File** button on **TmtEditor Toolbar;**
- Enter the name for the new file in the popup **File Browse Window**
- Select the template file to use for the new file in the popup **Template Selection Window** as shown bellow, and click on the **OK button.**



Browse in a TMT file

The data in a TMT file can be browsed in the **Variable Browser.**

- The first item in the browser is ROOT item, which is always there and cannot be edited
- A '+' or a '-' sign ahead of a variable indicates that it is a structure. Double click on a structure will expand or shrink it in the browser
- When a variable is selected, details on the variable will be listed in the **Variable Editor** to the right of the browser

Enable/disable template editing

By enabling template editing, all properties of all variables in the TMT file can be edited. If template editing is disabled, only the values of the variables given in the file can be changed. This allows data to follow exactly the format in the original TMT file (or a template file).

To enable or disable template editing, select **Enable/Disable template editing** from **Option Menu;**

Edit Variables

Add a numeric variable

- Select a structure variable in the **Variable Browser** and click on the **NUM** button to add a numeric variable under the structure selected.
- The new variable will be named 'new_num' and has a default value of being an empty matrix. The name and value for the new variable can be modified in the **Variable Editor.**

Add a character variable

- Select a structure variable in the **Variable Browser** and click on the **CHA** button to add a character variable under the structure selected.
- The new variable will be named 'new_char' and has a default value of being an empty string. The name and value for the new variable can be modified in the **Variable Editor.**

Add a structure variable

- Select a structure variable in the **Variable Browser** and click on the **Struct** button to add a structure variable under the structure selected.
- The new variable will be named 'new_stru', which contains an empty numeric variable 'new_num' and an empty string 'new_char', The new structure and the variables under the structure can be modified in the **Variable Editor.**

Move up/down a variable

- Select a variable in the **Variable Browser** and click on the **Up** or **Down** button to move a variable up or down in the TMT file

Delete a variable

- Select a variable in the **Variable Browser** and click on **Delete** button to remove the variable

Note: If a structure is to be deleted, all "children" items under the structure will also be deleted; if the numeric variable or the string variable to be deleted is the last child of a structure, the structure will be deleted along with the variable. The only exception is for the ROOT item, which can never be deleted.

Change the name of a variable

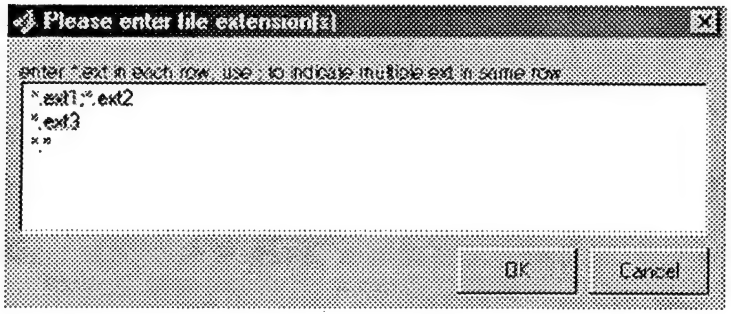
- Change the name in the **Name input box** on the **Variable Editor**
- Click on the **Save** button on the **Variable Editor**

Note: Array of structure is supported. The index of a structure is indicated by following the name of the structure by the index in the brackets

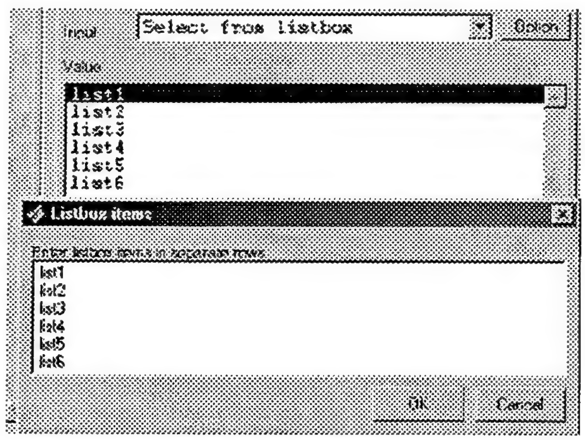
Change the input method of a string variable

- Select the input method in the **Input Method Selection Box** on the **Variable Editor**
- If the input method is **"browse for a file",** a **Browse** button will appear. Click on the button to select or enter the name of the file. The file name will be used as the value of the string variable.

Click on the **Option** button to change the file extension to be used for browsing files. The following is an example of how file extension is specified



- If the input method is **"Select from listbox",** the value of the string should be selected from the list box underneath the **Input Method Selection Box.** Click on the **Option** button the items to be listed in the listbox.



Add heading to a TMT file

To add or modify the heading a TMT file, click on the **Heading/Comments** button on the **Editor View Panel.** And enter the new heading in the popup window.

Get Quick Help

To get quick help and information about **TmtEditor,**

- Push **Show Help** button on **TmtEditor Viewbar;**

  Or slect **Help TmtEditor** from **Help Menu**

- Select a topic to display the help message about the topic

B. XY Plot Viewer

Introduction

What is XY plot viewer

XYViewer is graphical viewer to visualize the vectors or matrices stored in JIF ASCII files or STDMAT binary files.

Features

- Support STDMAT and JIF ASCII file formats, where the value of each "channel" of data is stored in a column vector and associated with a *name, label, units.* In STDMAT files, the data channels can also be grouped under different *group names;*
- Multiple data files can be opened in the same viewer. This allows the comparison of data from different files;
- Support print the plot as PS, EPS, EMF, and BITMAP files;
- Support output plot data as space, tab or comma delimited ASCII files;
- A simple GUI layout allows the easy access and plotting of data;
- Support overlay of curves on a single plot;
- Easy access to data definition and peak values;
- Many axis properties, such as color, grid, box, legend, axis label, title, can be edited;
- Many line properties ,such as style, width, color, marker and marker size, can be edited;
- Automatically synchronized with additional data viewer, such as **stickViewer,** to visualize data on fly.

Start XY Plot Viewer

XYViewer is delivered in one of the following three versions

- **MEX version**: MEX version of **XYViewer** is to be used in Matlab environment. To start it in Matlab, type '*xyviewer*' in Matlab command window.

Note: To use XYViewer in Matlab, the right path has to be setup for the directory where XYViewer MEX routines are installed

- **Standalone version:** Standalone version of **XYViewer** is delivered in a single installation file **"install_xyviewer.exe"**, which can be installed and run as a standard DOS/Window executable program.

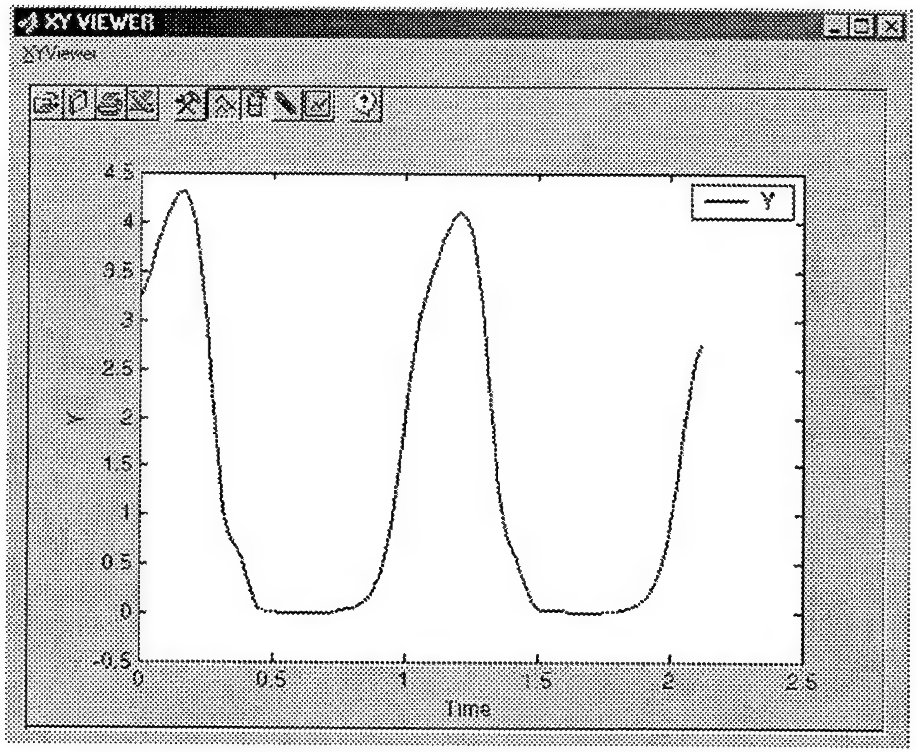
- **Application version: XYViewer** can also be integrated as part of application software as a data viewer. In this case, it can only be used with the software.

GUI Components

This section describes several GUI components of **XYViewer** and their common use.

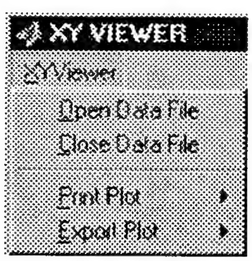
XY Plot Axis

XY Plot Axis is where the data is plotted.



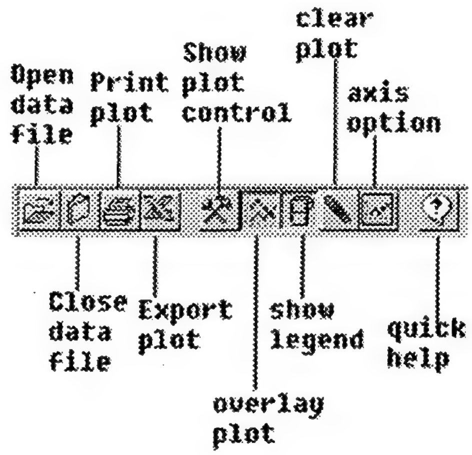
XYViewer Menu

The **XYViewer Menu** performs file operations, such as opening and closing of data files, as well as printing and exporting of a plot.



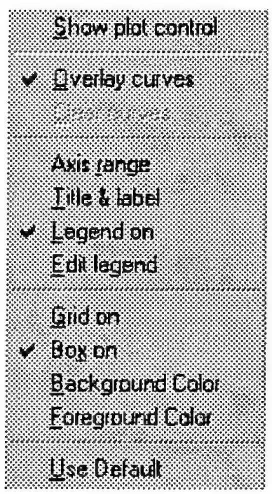
XYViewer Toolbar

XYViewer Toolbar provides easy access to common operations:



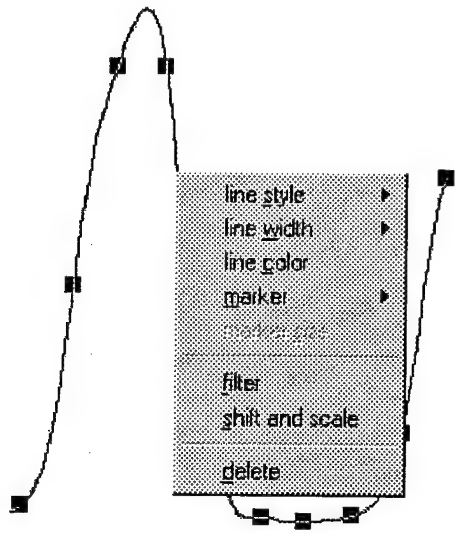
Axis Property Context Menu

Axis Property Context Menu is launched by right clicking mouse inside the **XY Plot Axis** but not over any curve plotted inside. It provides options to edit the properties of the axis.



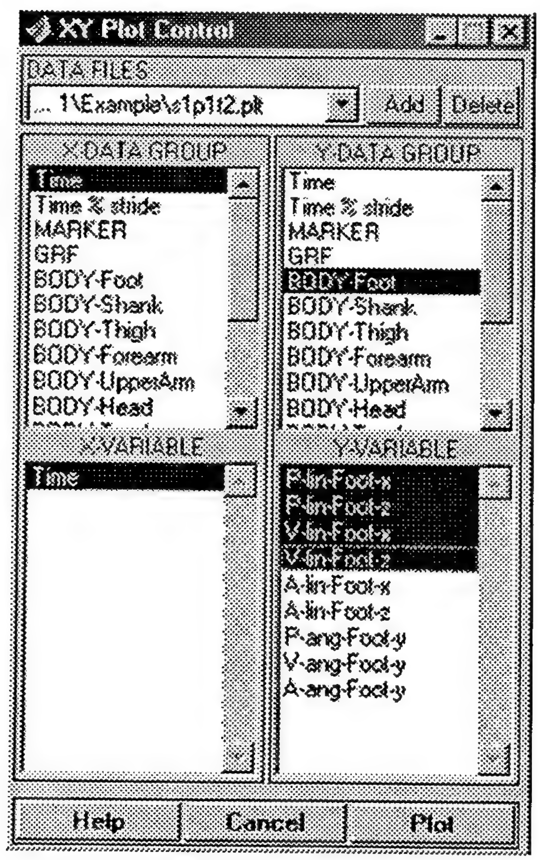
Line Property Context Menu

Line Property Context Menu is activated by right clicking mouse over a curve inside the **XY Plot Axis**. It provides options to edit line properties of the curve selected.



XY Plot Control

XY Plot Control is a popup window that displays the file(s) being opened and data inside a selected file. It can be used to open or close data file(s) and to plot data.



Use XY Plot Viewer

Open/Close XY Data File

To open a data file, follow the following steps

- Select **Open Data File** from **XYViewer Menu;**

  or push the **Open Data File** button on **XYViewer Toolbar;**

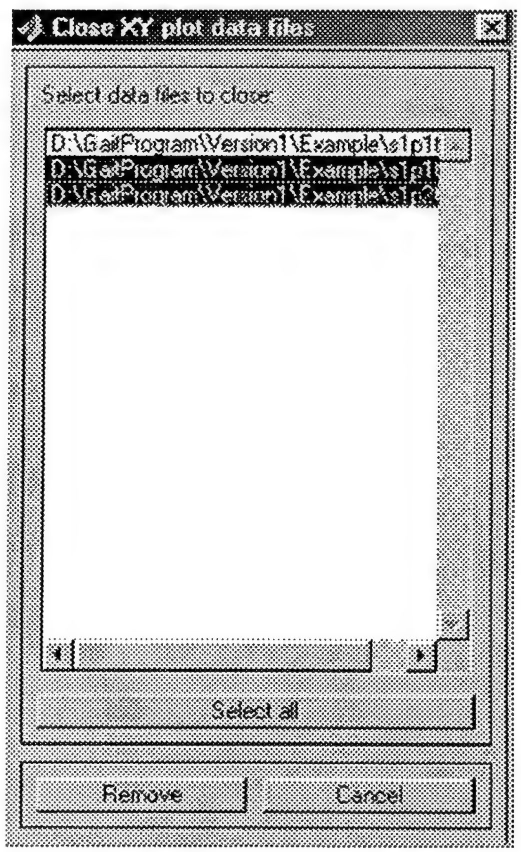
  or push the **Add** button inside **XY Plot Control**
- Browse for the file to open in the popup **File Browse Window**

To close a data file, follow the following steps

- Select **Close Data File** from **XYViewer Menu;**

  or push the **Close Data File** button on **XYViewer Toolbar;**

  or push the **Delete** button inside **XY Plot Control**
- Select the file(s) to close in the popup **Close XY plot data files** window and click **Remove** button to close the data file(s).



Print/Export a Plot

To print a plot on the **XY Plot Axis,** follow the following steps

- Select **Print Plot** from **XYViewer Menu;**

  or push the **Print** button on **XYViewer Toolbar;**
- Select **Default Printer** to print to the default printer
- Select the **type of file** and enter the file name in the popup **file browser window** to print the plot as a file. Current the following types of files are supported
  - Postscript files(*.ps)
  - Encapsulated postscript files (*.eps)
  - Window meta files (*.emf)
  - Bitmap files (*.bitmap)

To export the curves plotted in the **XY Plot Axis** to an ASCII file, follow the following steps

- Select **Export Plot** from **XYViewer Menu;**

  or push the **Export** button on **XYViewer Toolbar;**
- Select the **type of file** and enter the file name in the popup **file browser window** to export the data. Current the following types of files are supported
  - Space/tab delimited ASCII (*.dat)
  - Comma delimited ASCII files (*.csv)
  - JIF ASCII files (*.jif)

Plot Data

Following the following steps to plot data

- Activate the **XY Plot Control.** This can be done by doing one of the following:
  - Push **Show XY** button on **XYViewer Toolbar;**
  - In **Axis Property Context Menu** (by right click mouse in the **XY Plot Axis**), select **Show Plot Control;**
  - Push **Axis Properties** button on **XYViewer Toolbar** and select **Show Plot Control;**
- Select the right data file in the **Data File** popup window
- In **X Data Group** listbox, select the group name of X-axis variable
- In **X Variable** listbox, select the X-axis variable
- In **Y Data Group** listbox, select the group name of Y-axis variable(s)
- In **Y Variable** listbox, select the Y-axis variable(s)

- Click the **Plot** button to plot the selected Y-variable(s) vs. X-variable.

Edit Axis Properties

- Activate the **Axis Property Context Menu** by doing one of the following:
  - Push **Axis Properties** button on **XYViewer Toolbar;**
  - Right click mouse in the **XY Plot Axis.**
- Select one of the following properties to edit
  - **Axis range**
  - **Title & Label**
  - **Legend on**
  - **Edit legend**
  - **Grid on**
  - **Box on**
  - **Background color**
  - **Foreground color**

Edit Line Properties

- Activate the **Line Property Context Menu** by right clicking mouse over the curve whose properties is to be edited
- Select one of the following properties to edit
  - **Line style**
  - **Line width**
  - **Line color**
  - **Marker**
  - **Marker size**

Show/Hide Legend

The legend for the curves plotted can be displayed or removed from **XY Plot axis** by one of the following

- Push **Legend on/off** button on **XYViewer Toolbar;**
- Or select **Legend on/off** in **Axis Property Context Menu** (by right click mouse in the **XY Plot Axis**)

Enable/Disable Overlaying Plots

Overlaying plot can be enabled or disabled by one of the following

- Push **Overlay curve** button on **XYViewer Toolbar;**
- Or select **Overlay curves** in **Axis Property Context Menu** (by right click mouse in the **XY Plot Axis**)

Clear a Plot or Delete a Curve

One of the following will clear the plot

- Push **Clear** button on **XYViewer Toolbar;**
- Or select **Clear curves** in **Axis Property Context Menu** (by right click mouse in the **XY Plot Axis**)

To delete only one curve from the plot, follow the following steps

- Activate the **Line Property Context Menu** by right clicking mouse over the curve whose properties is to be deleted
- Select **Delete**

Get Quick Help

To get quick help and information about **XYViewer,**

- Push **Help** button on **XYViewer Toolbar;**
- Select a topic to display the help message about the topic

C

C. Stick Plot Viewer

Introduction

What is Stick plot viewer

StickViewer is a three-dimensional viewer to visualize stick plots.

Features

- Simple GUI provides easy access to visualization and animation
- Support print the plot as PS, EPS, EMF, and BITMAP files;
- Support output stick animations as Matlab movies;
- Many axis properties, such as color, grid, box, legend, axis label, title, can be edited;
- Many stick properties ,such as style, width, color, marker and marker size, can be edited;
- Automatically synchronized with **XYViewer** to visualize data on fly.

Start STICK Plot Viewer

StickViewer is delivered in one of the following three versions

- **MEX version:** MEX version of **StickViewer** is to be used in Matlab environment. To start it in Matlab, type *'stickviewer'* in Matlab command window.

Note: To use **StickViewer** in Matlab, the right path has to be setup for the directory where **StickViewer** MEX routines are installed

---

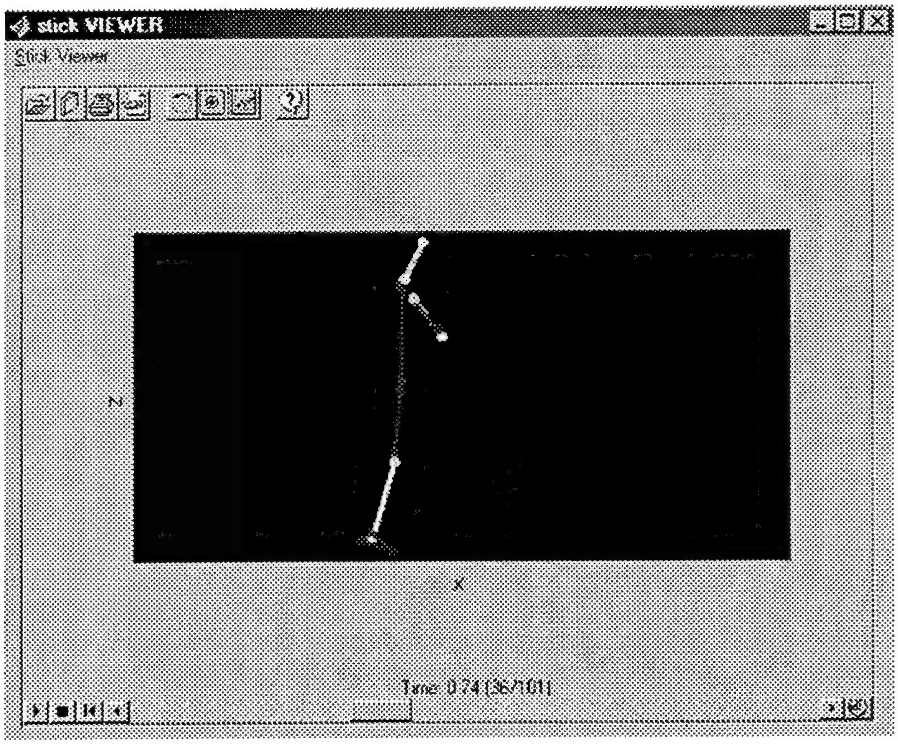
- **Standalone version:** Standalone version of **StickViewer** is delivered in a single installation file **"install_stickviewer.exe"**, which can be installed and run as a standard DOS/Window executable program.
- **Application version: StickViewer** can also be integrated as part of application software as a data viewer. In this case, it can only be used with the software.

GUI Components

This section describes several GUI components of **StickViewer** and their common use.

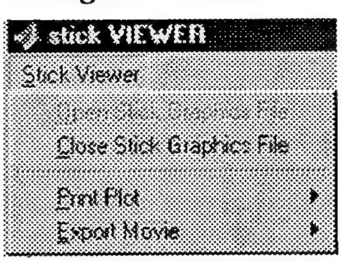
Stick Plot Axis

Stick Plot Axis is where the data is plotted.



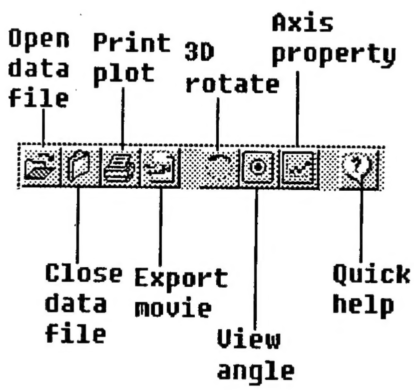
StickViewer Menu

The **StickViewer Menu** performs file operations, such as opening and closing of data files, as well as printing plot and exporting animation.



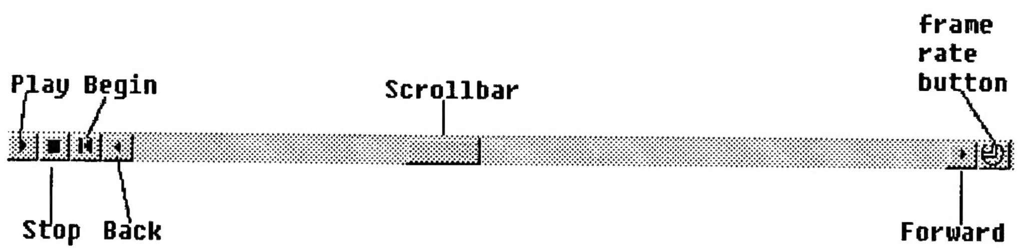
StickViewer Toolbar

StickViewer Toolbar provides easy access to common operations:



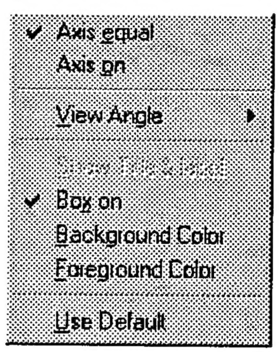
Stick Animation Toolbar

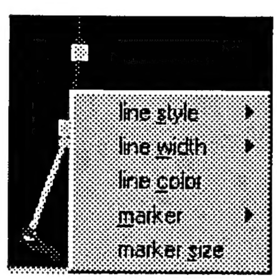
Stick Animation Toolbar provides options to control the animation of stick plots.



Axis Property Context Menu

Axis Property Context Menu is launched by right clicking mouse inside the **Stick Plot Axis** but not over any stick. It provides options to edit the properties of the axis.





Stick Property Context Menu

Stick Property Context Menu is activated by right clicking mouse over a stick inside the **Stick Plot Axis**. It provides options to edit line properties of the stick selected.

Use STICK Plot Viewer

Open/Close Stick Graphics File

To open a data file, follow the following steps

- Select **Open Stick Graphics File** from **StickViewer Menu;**

  or push the **Open** button on **StickViewer Toolbar;**

- Browse for the file to open in the popup **File Browse Window**

- The first frame of the stick plots will be displayed

To close a stick graphics file, follow the following steps

- Select **Close Stick Graphics File** from **StickViewer Menu;**

  or push the Close button on StickViewer Toolbar;

Note: Stick graphic file is a Matlab binary data file containing the time history of positions and orientations of each stick, as well as the definition of each stick

---

Print a Frame

To print a snapshot of a stick plot frame the following steps

- Select **Print Plot** from **StickViewer Menu;**

  or push the **Print** button on **StickViewer Toolbar;**

- Select **Default Printer** to print the snapshot to the default printer

- Select the **type of file** and enter the file name in the popup **file browser window** to print the snapshot as a file. Currently the following types of files are supported

  - Postscript files(*.ps)

  - Encapsulated postscript files (*.eps)

  - Window meta files (*.emf)

  - Bitmap files (*.bitmap)

Export an Animation

To export a stick animation, following the following steps

- Select **Export Movie** from **StickViewer Menu;**

  or push the **Export** button on **StickViewer Toolbar;**

- Select the **type of file** and enter the file name in the popup **file browser window** to export the data. Currently only Matlab movie (*.mat) is supported

Note: To convert a Matlab movie file into an AVI file, use *"movie2avi"* command in Matlab

Edit Axis Properties

- Activate the **Axis Property Context Menu** by doing one of the following:
  - Push **Axis Properties** button on **StickViewer Toolbar;**
  - Right click mouse in the **Stick Plot Axis.**
- Select one of the following properties to edit
  - **Axis equal:** force all axes to use the same data aspect ratio for
  - **Axis on:** turn on or turn off axis
  - **View angle:** change the view angle of the 3D stick plots
  - **Show Title & lable:** show label for the axes
  - **Box on:** add or remove box from the axis
  - **Background color:** change background of the axis
  - **Foreground color:** change foreground of the axis

Edit Stick Properties

- Activate the **Stick Property Context Menu** by right clicking mouse over the stick whose properties is to be edited
- Select one of the following properties to edit
  - **Line style**
  - **Line width**
  - **Line color**
  - **Marker**
  - **Marker size**

Free rotation of Axis

The free rotation of the axis (to change view angle) can be performed by pushing down the **3D rotate** button on **StickViewer Toolbar** and use mouse to rotate the **Stick Plot Axis.**

Get Quick Help

To get quick help and information about **STICKViewer,**

- Push **Help** button on **STICKViewer Toolbar;**
- Select a topic to display the help message about the topic